

PAD-BASED: PROTOTYPES AND DELEGATION  
BASED APPROACH TO KNOWLEDGE  
ORGANIZATION IN EXPERT  
SYSTEM DESIGN

By

JARERNSRI LIMSUPAVANICH-MITRANONT

Bachelor of Science (Physics)  
Mahidol University  
Bangkok, Thailand  
1980

Master of Science (Applied Mathematics)  
Mahidol University  
Bangkok, Thailand  
1983

Submitted to the Faculty of the  
Graduate College of the  
Oklahoma State University  
in partial fulfillment of  
the requirements for  
the degree of  
DOCTOR OF PHILOSOPHY  
December, 1993

PAD-BASED: PROTOTYPES AND DELEGATION  
BASED APPROACH TO KNOWLEDGE  
ORGANIZATION IN EXPERT  
SYSTEM DESIGN

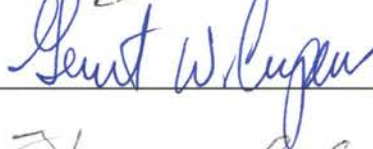
Thesis Approved:

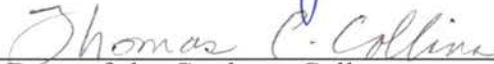
  
\_\_\_\_\_

Thesis Adviser

  
\_\_\_\_\_

  
\_\_\_\_\_

  
\_\_\_\_\_

  
\_\_\_\_\_

Dean of the Graduate College

## ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my dissertation advisor, Dr. K.M. George, for his continual guidance, support, dedication, kindness, and giving invaluable time and instruction toward this dissertation. With his encouragement and motivation, this dissertation has been accomplished.

My sincere gratitude is due to my advisory committee chairman, Dr. G.E. Hedrick, for his support and guidance particularly on my initial work. Special thanks are due to Dr. H. Lu for her support, kindness and understanding throughout the completion of this work, and also to Dr. P. Benjamin for his valuable suggestions as an expert in the field. Also my deepest gratitude is expressed to Dr. G.W. Cuperus who supervises, inspires, and supports me both emotionally and financially while I worked with him for five years, and Dr. R. Wright, Entomology Department, for the assistantship. I am grateful to Dr. J. Stritzke, Agronomy Department, for his valuable time and expertise, to all Computer Science staffs and Entomology staffs, and to Prof. H. E. Kaiser for the English review.

Finally, my heartfelt love and appreciation are expressed to my beloved husband, Nakul Mitranont, my dearest son, Toffy, for their invaluable mental and emotional motivations and support which have been the driving force for this achievement. My love and heart belong to my 88-year old grandmother, to my father who dedicated his life raising and educating his six children, and especially to my mother in heaven who passed away in March 13, 1990 while I was working on this dissertation. My extended love and thanks are expressed to my brothers, sisters and their family particularly Jaroenporn and Ekkehard Betsch for their support.

## TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION.....	1
II. LITERATURE REVIEW.....	3
2.1 Expert Systems.....	3
2.1.1 A Simple Expert System Architecture.....	4
2.2 Object-Oriented Systems.....	5
2.2.1 Inheritance and Set Approach.....	7
2.2.2 Delegation and Prototype Approach.....	8
2.2.3 Polymorphism in Object-Oriented Systems.....	11
2.3 Object-Oriented Approaches and Expert Systems.....	12
III. PROBLEM STATEMENT.....	15
3.1 Common Problems in Expert System Design.....	15
3.2 Objectives.....	16
IV. PAD-BASED: Prototypes and Delegation-Based Approach to Knowledge Organization in Expert System Design.....	18
4.1 Prototype and Delegation Conceptual Model.....	18
4.2 PAD-BASED Expert System Architecture.....	19
4.2.1 Prototype Basic Elements.....	21
4.2.2 Interaction among Prototypes.....	23
V. IMPLEMENTATION OF THE PAD-BASED MODEL.....	26
5.1 Agricultural Applications.....	26
5.2 PAD-BASED Expert System Development Life-Cycle.....	30
5.3 WEEDPLUS: Weed Management Expert System.....	38
5.4 Designing WEEDPLUS Expert System.....	38
5.4.1 Task Organization.....	39
5.4.2 Prototypes and Interfaces Construction.....	41
5.5 Interactions among Prototypes in WEEDPLUS.....	44

VI. ADVANTAGES OF THE PAD-BASED MODEL.....	46
6.1 Reducing Development Time.....	46
6.2 Providing an Alternate Memory Management Scheme for Small Computer System.....	48
6.3 Increasing Knowledge Sharing and Re-use.....	54
6.3.1 Internal Knowledge Sharing and Re-use.....	54
6.3.2 External Knowledge Sharing and Re-use.....	59
6.3.2.1 PROFALF--an Expert System to Estimate Profitability of Alfalfa.....	59
6.3.2.2 Incompatible Structure.....	62
6.3.2.3 Compatible Structure.....	64
VII. CONCLUSIONS AND FUTURE WORK.....	66
BIBLIOGRAPHY.....	69
GLOSSARY.....	75

## LIST OF FIGURES

Figure	Page
1. A Simple Expert System Architecture.....	5
2. Prototypes and Delegation Conceptual Model.....	19
3. A PAD-BASED Expert System Architecture.....	20
4. Basic Elements of a Prototype.....	22
5. Example of Prototypes in PAD-BASED System.....	24
6. Delegation Mechanism and Message Passing.....	25
7. A Simple Conceptual Structure for Expert System Development.....	31
8. The PAD-BASED Expert System Development Structure; Tii represent a Task; Pii represents a Prototype.....	33
9. The PAD-BASED Expert System Development Life Cycle.....	34
10. System Development Phase in the PAD-BASED Development Life Cycle	36
11. Organization of the Main Tasks in WEEDPLUS.....	39
12. Subtasks Organization in Weed Management, Weed Information, and Herbicide Information.....	40
13. Prototypes Construction Corresponding to Task Organization.....	41
14. Example of Prototypes in WEEDPLUS.....	43
15. Delegation Mechanism in WEEDPLUS.....	45
16. Memory Organization in WEEDPLUS.....	49
17. Active and Inactive Prototypes during the Consultation.....	52

Figure	Page
18. Internal Knowledge Sharing and Re-use in WEEDPLUS.....	55
19. Modification of TASK 2 after Adding INDIANA Weeds.....	56
20. Prototypes Construction after Adding Indiana Weed Knowledge; Indiana Farmers Reuse Knowledge in WEEDPLUS.....	57
21. Modification in the Interface after Adding a New Prototype.....	58
22. Knowledge Re-use in WEEDPLUS via Delegation Mechanism.....	58
23. Components of the AIM System.....	60
24. PROFALF Schematic Diagram.....	61
25. Subsystem Organization of PROFALF.....	61
26. Conceptual Idea for Non PAD-BASED Expert System to Reuse Knowledge from a PAD-BASED Expert System.....	62
27. The General Structure of the Delegator Prototype.....	63
28. Knowledge Sharing and Re-use between Two PAD-BASED Expert Systems in Macro Level.....	64
29. Knowledge Sharing and Re-use in Micro Level between Two PAD-BASED Expert Systems.....	65

## CHAPTER I

### INTRODUCTION

In the past decade, the object-oriented approaches set a new direction in most computer research and application areas such as programming languages, artificial intelligence, databases, and distributed systems. This rapid growth is attributed to its four significant properties: the principles of program modularity, information hiding, data abstraction, and code reusability. Studies of the contributions of the object-oriented approach are prolific such as in [Zhu et al. 91], [Alpert et al. 90], and [Ramamoorthy 88]. The objectives of this research are to apply the object-oriented approach to expert systems, and to develop a new flexible model (PAD-BASED model) based on the concept of prototype and delegation mechanism proposed as an alternative architecture to designing, maintaining and/or interfacing expert systems in both macro and micro levels.

This new approach is derived to overcome the problem of lengthy development time in expert system design, limited knowledge sharing and re-use in expert system applications [Mitrpanont et al. 94], developing an expert system in a small computer system with restricted internal memory [Mitrpanont et al. 94], and complexity in expert system maintenance. This approach is applied to the problems in agricultural applications. This application area is broad and significant for human life. Therefore, expert system techniques have been used successfully as a key tool for developing and delivering



knowledge to the farmers. However, the main problems in this area are the lengthy development time, limited knowledge sharing among the experts and re-use, and the limited distribution among the major users (i.e., the farmers).

In general, knowledge in a PAD-BASED expert system is organized as a collection of prototypes. The interactions among these prototypes utilize the delegation and simple message passing mechanisms. Prototype structure is used because it increases structurability and modularity without the class-subclass complexity. Delegation and simple message passing mechanisms play the significant role to increase knowledge sharing and reusability in PAD-BASED expert system. As stated by Lieberman [Lieberman 86a], prototype and delegation mechanism provide high opportunity on knowledge sharing. Furthermore, delegation mechanism is also the key to organize internal memory usage in a small computer during the consultation process or during the interactions among prototypes.

## **CHAPTER II**

### **LITERATURE REVIEW**

#### **2.1 Expert Systems**

Expert system research has been a prominent field in artificial intelligence (AI). It has been used as a new set of tools to develop applications in business, education, industry, and government areas. Furthermore, it is a special tool to integrate different types of knowledge and to deliver knowledge from the experts to the common users who need but do not possess the knowledge for problem solving and decision support.

As defined by Martin and Oxman, an expert system is a computer-based system that used knowledge, facts, and reasoning techniques to solve problems that normally require the abilities of human experts [Martin and Oxman 88]. As a result, expert system is a necessary tool to provide knowledge for the users who seek for the advice from the human experts. Currently there are variety of expert system development tools available as an alternative to programming languages such as LISP, PROLOG, and SMALLTALK. These tools are easier to use and provide more flexibility on different methods for knowledge representations and inference strategies. Knowledge engineer can select the particular knowledge representation method for the specific problem. These knowledge representation methods generally include rule-based system, structure-based system, frame-based system, and shell-based system. For examples, ROSIE [Klahr and Waterman 86]

is a programming environment for expert systems which integrates two programming paradigms--rule-based modeling and procedure-oriented computation. KEE [Kehler 84], Knowledge Engineering Environment, integrates frame-based and rule-based systems. LOOPS, by Xerox Corporation, includes data, object, rule-based and procedure-oriented programming languages (Interlisp-D: a dialect of the LISP programming language) [Bobrow and Stefik 83]. The systems that provide more than one way of knowledge representations are called hybrid environments. In addition, they also offer more features on different inference engines, several user interfaces including interface to databases, spreadsheets, graphics, hypermedia, and multimedia. In general, these hybrid systems are outstanding and very convenient for expert system design and development. However, users always pay the price by investing excessive money and time to learn how to use them and even more time to develop an application in a new environment. Hardware support for these systems is also another problem since some of them require a larger memory system, higher processing speed, or more specific devices such as mouse or a high resolution color monitor.

### **2.1.1 A Simple Expert System Architecture**

In the conventional expert system, there are three main parts: user interface, knowledge database, and inference engine as shown in figure 1 [Martin and Oxman 88]. The user interface allows the knowledge engineer to enter knowledge as rules and facts into the system, allows the user to query and obtain knowledge from the system, and supports several communication (interface) between users and the system.

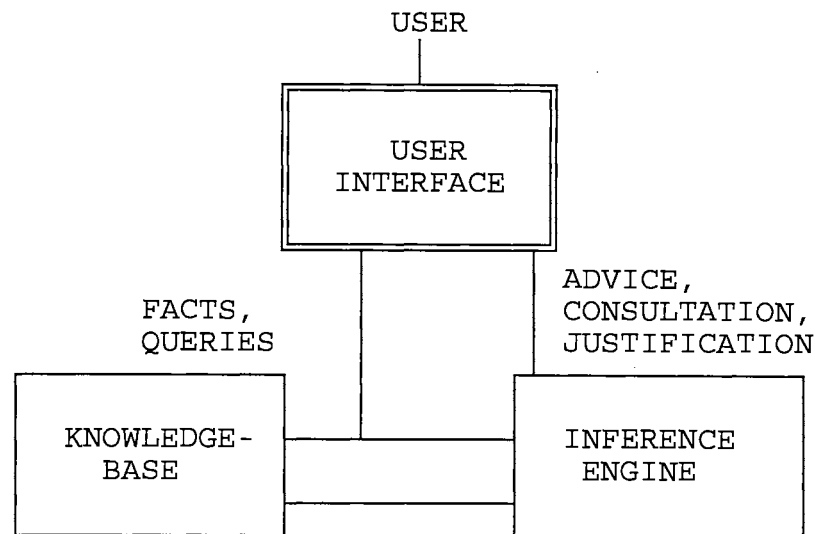


Figure 1. A Simple Expert System Architecture (Source: Martin and Oxman, "Building Expert Systems", Prentice Hall, Englewood Cliffs, NY, 1988, p. 26)

The knowledge-base contains the knowledge obtained from experts. This knowledge is usually represented in some particular format depending on the structure used. The knowledge representation method must be compatible in format to allow the inference engine to access. The inference engine provides the inference strategy to access the knowledge obtained from the user and knowledge from expert stored in the knowledgebase to infer some new facts. In other words, inference engine performs the deduction mechanism to generate some advice for the user by using the specific knowledge and information.

## 2.2 Object-Oriented Systems

In the conventional object-oriented approach, program consists of a collection of fundamental elements called objects. Each object has a unique identity, contains a private data structure, and can be accessed or modified by its predefined methods upon receiving

a proper message. The data structure is invisible to any other objects. In other words, the object maintains the principle of information hiding through the encapsulation concept which prohibits direct access to the private data and operations of each object. Operations define the behavior of an object in two aspects: 1) how it behaves in response to receiving a message from an external object; 2) how it controls the internal state transitions [Zhu et al. 91]. In general, an operation consists of a definition and an implementation. The definitions of the operations form an object's public interface which are accessible to other objects. Changes in implementations do not affect how the object is used. The internal data of an object is accessed via a set of message selectors stored in a public interface or protocol. Upon receiving a validated message, an object performs an operation by some internal state transitions or by initiating new messages to other objects including itself. This information hiding concept also improves data security and integrity.

Objects are grouped into classes or types according to their structure and behavior. In other words, objects with the same data structure and public interface (or same set of operations) belong to the same class or type. Classes are organized in a hierarchical manner in which the data structures and operations from the superclass are inherited to the subclasses. A mechanism called inheritance implements this feature. It takes the advantage of the set inclusion of objects' data structures and operations in superclasses and subclasses to provide code reusability.

There are three major mechanisms used in object-oriented systems: Inheritance, Delegation, and Polymorphism. However, inheritance and delegation have been studied as an alternative to each other such as in [Lieberman 86a] and [Stein 87]. The next three

sections describe the characteristics of these three mechanisms.

### **2.2.1 Inheritance and Set Approach**

Inheritance is an important mechanism for sharing behavior among objects in conventional object-oriented programming languages. It represents concepts as abstract sets or classes. By constructing the concept of the set, the objects or instances are classified by their common data structures and operations (or methods or behaviors) into classes or types. An object is created to represent the description (or common data structures and operations) of the class. Operation definitions form a class public interface which are accessible to other objects. A class body specifies code for implementing the operations defined in the class interface. Each class can have one or more interfaces. A new subclass can be generated by a class to extend additional data and behavior. These classes and subclasses are organized in a class hierarchy. Therefore, the behavior sharing is inherited through this classification from a superclass to subclasses via the inheritance mechanism. The principle of inheritance [Zhu et al. 91] states that:

*A class is defined to be a subclass of an existing class, then any data structure or operation in the superclass is also defined in the subclass. Furthermore, the definitions of the inherited data or operations are unchanged unless they are explicitly overwritten.*

Inheritance from a single superclass is called a single inheritance while inheritance from multiple superclasses is called multiple inheritance. Basically, operation redefining (in single inheritance) and conflict resolution (in multiple inheritance) by subclasses are error prone because semantic inconsistencies are difficult to detect and complicated to control.

This difficulty is the most controversial problem in inheritance. In addition, inheritance requires two different kinds of interface between objects: subclass interface and instance interface.

### **2.2.2 Delegation and Prototype Approach**

A delegation mechanism is proposed as an alternative to inheritance. It is said to be a generalization of inheritance [Wegner 87]. In a system using delegation, objects are not represented in terms of a class or subclass but as a concrete prototype containing its own data structures and methods. By defining a prototypical object to represent individual concepts, a new object (called an extension object) is defined by the concepts that are different or variable from the prototype [Lieberman 86a]. This new object can also reuse part of the knowledge in a prototype. In addition, every object in this system can behave as a prototype for the creation of a new object. Objects share the common behavior using the delegation mechanism. Delegation removes the distinction between classes and instances. An object forwards or delegates a message to another object (called the prototypical object) to execute the operation corresponding to that message. All the possible paths of delegation must be specified in an extension list. In other words, an extension object has a list containing all its possible prototypes and a part containing its own personal data. When an extension object receives a message, first it checks its own personal data. If it cannot respond to that message, then it forwards the message to its prototypes on the list to find one that can respond to the message. Furthermore, when a delegated object receives a message, it also obtains a component called the client which specifies the object originally receiving the message. Thus, the delegated object is able

to identify this specific object that needed the operation.

Because any object can be defined as a prototype, and any message can be forwarded at any time, Lieberman specifies that delegation is more flexible than inheritance in which it is more powerful for combining behaviors from multiple sources and it is advantageous for highly interactive and incremental software development.

In addition to Lieberman, others [Aksit et al. 91] agree that delegation is very useful in building extensible and open systems. Borning [Borning 86] uses prototypes as an alternative to resolve the complexity of classes and metaclasses in Smalltalk. Instead of performing a class-subclass initiation, a new object is created by simply copying or cloning from a prototype. He introduces the concept of using inheritance constraints as the object hierarchies to manipulate inheritance relations. He also mentions that prototypes are often used in visual programming, and artificial intelligence representing languages. Wegner [Wegner 87] gives a broader concept of delegation as one of the six orthogonal dimensions of object-oriented language design. A general term, class-independent, is used to define delegation for dynamic hierarchical resource sharing. To use class as an orthogonal dimension that spans the object-oriented design space, delegation is determined as a mechanism that allows objects to delegate responsibility for performing an operation, or finding a value, to one or more designated ancestors. Then, inheritance is viewed as a specialization of delegation in which the entities that inherit are classes. Ungar and Smith [Ungar and Smith 87] implement the concepts of prototypes and inheritance as a new object-oriented programming language called "Self" which uses slots to contain the variables and procedures. Almarode [Almarode 89] presents a mechanism to model the various semantics of delegation in which he believes it is one



drawback of delegation. The proposed mechanism called rule-based delegation uses rules to control the delegation of messages. It is used to implement classical single and multiple inheritances. Moreover, rules can be created dynamically to model application-specific semantics. In Johnson and Zweig [Johnson and Zweig 91], delegation is added into C++ language with the belief that it has advantages over inheritance. For instance, delegation simplifies the programming model; it is easier to implement the one-of-a-kind object and make programming more concrete; it is easier for objects to change their behavior.

In response to Lieberman, Stein [Stein 87] argues on behalf of inheritance that on a particular view of classes, inheritance can be used to implement delegation and its prototype-based systems. These two points of view indicate that inheritance and delegation are very little different in terms of implementation. However, much research has been done to define and to compare on these two controversial mechanisms. Tomlinson [Tomlinson et al. 89] characterizes object-oriented systems by the sharing protocol and the organization protocol. They present a detailed contrast of inheritance and delegation by analyzing the basic performance characteristics of simple delegation and inheritance protocols, and some optimizations of these protocols. They summarize that in most situations, inheritance has a better speed than delegation, and the speed and space optimization for inheritance usually outweigh the need of flexibility offer by delegation. However, they point out that:

*The prototype approach is more applicable in those cases where most objects have different structure or sharing requirements. Knowledge-based applications, in particular, exhibit these properties to a degree.*

Aksit [Aksit et al. 91] presents a technique called atomic delegation that allows an object to delegate a sequence of request messages to one or more designed objects as an atomic action. By changing the delegation relation between objects or by modifying the functionality of an object that the messages are delegated to, you can dynamically change the set of atomic actions supported by an object.

### **2.2.3 Polymorphism in Object-Oriented Systems**

Cardelli and Wegner [Cardelli and Wegner 85] define two kinds of polymorphism: universal polymorphism and ad hoc polymorphism. Universal polymorphism is a general term used to cover any mechanism which works on an infinite number of types. Ad hoc polymorphism covers techniques which work on a limited, specified set of types.

In object-oriented languages, there are two forms of polymorphism. The first form is concerned with typing; i.e. an object can have more than one type. Thus, a particular object can be manipulated by operations associated with any of those types. The second form is independent to type and the operations can be applied to distinct types which correspond to disjoint sets.

Based on Wegner's definition, Blair [Blair et al. 89] proposes a new definition for object-oriented systems that the systems should embody the notions of encapsulation, and set-based abstraction; and should also support two forms of polymorphism: inclusion polymorphism and operation polymorphism. Inclusion polymorphism implies that objects can belong to more than one set by set intersection. There are two aspects of inclusion polymorphism. First, as-if polymorphism features the objects that belong to one set but can be used as if they belong to another set through inclusion. Second, behavior sharing

features the objects that belong to one set but can share behavior (operations) of another set through the property of inclusion. Implicit behavior sharing occurs as a direct result of inclusion while explicit behavior sharing is explicitly described in the language or system.

Operation polymorphism deals with the operations with the same name that can be applied to different objects which have no relationship in terms of inclusion. These operations are interpreted in the context of the particular object. Two ways are used to achieve this feature: an ad hoc mechanism, and a form of universal polymorphism. In the first one, methods are overloaded on the same name but map to the different code bodies. The second method is called parametric polymorphism. Operations are accessible on a range of types implementing the same behavior irrespective to object type. They use this new paradigm to map inheritance and delegation mechanisms. They conclude that there are good alternatives to inheritance that provides more degrees of freedom in implementation especially in the context of distributed system architectures.

### **2.3 Object-Oriented Approaches and Expert Systems**

The impact of the object-based computation on expert systems is investigated by Ramamoorthy and Sheu [Ramamoorthy and Sheu 88.] Object-oriented programming is merged into AI programming languages such as Lisp, Prolog, Flavors, Loops, CommonLoops, and Concurrent Prolog. The availability of these tools provides two additional features: object abstraction for hierarchical reasoning, and expert cooperation for distributed problem solving. A typical example of integrating distributed experts to achieve a common goal is found in KEE [Fikes and Kehler 85] where rules are grouped

into classes, and invoked by methods. Loops [Stefik and Bobrow 86] is another language integrated rule-based and object-oriented approaches. Rules in Loops are grouped into rule sets, and invoked by message sending. Leonardi [Leonardi et al. 89] applies the concepts of prototypes and delegation to Prolog called the Prolog-Prototypes. The concepts such as modularity, information hiding, information sharing, and knowledge structuring are fundamental to the application domains of logic programming languages such as knowledge-based systems. Therefore, combining logic and object-oriented programming in Prolog-Prototypes provides a new tool that is very useful in building knowledge-based systems. Franke [Franke 90] defines an object-oriented protocol as a set of messages to perform both forward and backward chaining inferencing that allows the integration of an inferencing mechanism with CAD tools. Ibrahim and Woyak [Ibrahim and Woyak 90] use the EDS/OWL environment to define behaviors (methods, slot access, etc.) as objects adhering to a common protocol. Another approach asserts that a knowledge-based task can be accomplished by decomposing it into several agents; each contains and represents a chunk of the complete knowledge needed to accomplish the overall task [Alpert et al. 90]. This approach has been used to construct many AI systems including system development environments and domain-specific knowledge-based applications. Lieberman [Lieberman 87] points out that because objects encapsulate both state and behavior, and because they possess the inherent communication capabilities; therefore, an object-oriented approach is a natural candidate for the implementation of distributed, multiple-agent environments. Moreover, Barghouti and Kaiser [Barghouti and Kaiser 90] explore the use of object orientation in multiple-agent environments, and provide a central issue of how to support cooperation among a team of software

developers working on the same project. They propose a method using object-oriented database with a concurrency-control mechanism using the object-oriented representation semantics. Leung and Wong [Leung and Wong 90] present a new architecture for an expert system shell that enables the mixing of rules and procedures, allows the automatic extraction of data from a database system, and provides a fuzzy database query facility. They use an object-oriented approach for knowledge representations and inferences in expert system shell, and conclude that this approach improves the consistency, maintainability, and structurability of the knowledge base. Narayanan and Jin [Narayanan and Jin 91] propose an object-oriented expert diagnostic system approach for the domain of hardware fault location and diagnosis. This approach emphasis is on its architecture and the role of diagnosis rule-based reasoning designed to provide a precise and powerful diagnostic system. Polymorphism is also introduced into expert systems to improve expert system maintainability [Yen et al. 91]. Polymorphism allows object-oriented system to separate a generic function from its implementation. Thus, it is introduced into rule-based paradigm to separate a rule's function from its implementation details by replacing a rule's action with a generic operation.

## **CHAPTER III**

### **PROBLEM STATEMENT**

#### **3.1 Common Problems in Expert System Design**

From the previous studies, the common problems in expert systems involve their structures and complex nature which increase the difficulty in expert system maintenance and extension. There are many approaches and architectures designed to solve these problems. Presently, the main idea is to apply the object-oriented approaches and techniques on the concepts of encapsulation (i.e., information hiding), and program modularity (i.e., system structurability) to the problems in expert systems. Most of the emphasis is on designing new techniques, new languages, or new environments for designing new expert systems based on applying object-oriented approach to the knowledge representation techniques such as rule-based system and expert system shell. Therefore, using these new approaches usually requires new languages, new environments, or new developing tools which means that users need to obtain new products (or tools) and learn how to utilize them while wasting previous resources such as man-power, money, and time that were invested. Consequently, money is needed to buy new resources and tools, and to rebuild the application software. However, in the real world, users want services such as expertise knowledge, problem solving methodology, diagnostic results, planning, or minimal explanation from expert systems. Furthermore, there are

many applications that once expert systems have been developed for use, they are used again and again. Most of them only need to be maintained either by extending them for a wider scope, or by modifying them for better usage and results. However, in practice, modifying and extending an expert system may introduce new problems such as a system error, a complex and unstructured system, and an internal memory limitation problem in a small machine such as in a 640K microcomputer system ([Schwartz 87], [Baran 89], [Malloy 89], [Bertolucci 90]). Users must weigh between the cost of maintenance of existing systems and the cost of new machines and new development tools.

### **3.2 Objectives**

In the past decade, although expert system development tools and techniques have been prolific and each of them has its distinguishing features, it would be beneficial to the users if they could use a simple designing technique to design and develop an expert system. It would be beneficial if we could use the same technique to easily maintain and expand the expert systems without creating all the above problems.

In this dissertation, a new expert system architecture using the object-oriented approach based on prototype and delegation mechanism (the PAD-BASED model) is presented. There are three main reasons for using an object-oriented approach based on the prototype and delegation technique. First, the four features of object-oriented system: information hiding, program modularity, data abstraction, and code reusability are the major contributions to expert system design. Secondly, the prototype approach in knowledge organization increases the expert system modularity and structurability which obviously simplifies the expert system maintenance. Finally, the delegation mechanism

is dynamic, more powerful, and more flexible to apply to the expert systems. The PAD-BASED model is used as a knowledge organization technique in expert system design which simplifies the developing process because knowledge representation can be any technique. It is demonstrated that using this model can help solving the above problems. Moreover, it also provides more advantages to simplify expert system maintainability and also support knowledge sharing and re-use.

The objectives of this dissertation are the following:

- 1) to provide a detailed description of the PAD-BASED model presented as a new architecture for knowledge organization technique in expert system design using the prototype and delegation approach;
- 2) to use this architecture as a new technique for designing an expert system;
- 3) to apply the PAD-BASED model to design and develop an expert system in agricultural application;
- 4) to demonstrate the advantages of the PAD-BASED model as a simple and powerful technique in designing and maintaining an expert system that helps increase its structurability and modularity, reduce its designing and development time, support knowledge sharing and re-use, provide an alternative to memory organization in small computer implementation.



## CHAPTER IV

### PAD-BASED:

#### PROTOTYPE AND DELEGATION-BASED APPROACH TO KNOWLEDGE ORGANIZATION IN EXPERT SYSTEM DESIGN

##### 4.1 Prototype and Delegation Conceptual Model

PAD-BASED is an acronym for a new architecture presented as a knowledge organization technique in expert system design based on the concepts of prototype and delegation approach. Figure 2 depicts the conceptual model of the prototype and delegation mechanism. In this system, knowledge is organized as a prototype. New prototypes can *re-use* part or all of the knowledge stored in or derived from the shared behaviors of the prototype. The mechanism to utilize the shared knowledge and behaviors of the prototype is called delegation mechanism. Prototype P1 sends a message to prototype P2. P2 checks its data structures and behaviors and finds that it cannot answer to that message. Therefore, P2 delegates the message to prototype P3 which contains the behavior corresponding to that message. P3 also obtains the knowledge from the delegator, P2, and uses this knowledge for the operation. Finally, P3 sends the response back to the object, P1, that initiated the message in the first place via a simple message passing.

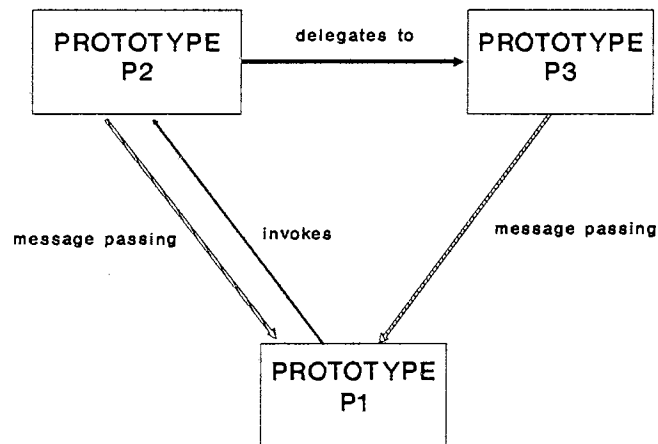


Figure 2. Prototypes and Delegation Conceptual Model.

#### 4.2 PAD-BASED Expert System Architecture

In general, an expert system is defined as a computer-based system that uses knowledge, facts, and reasoning techniques to solve problems that normally require the abilities of human experts [Martin and Oxman 88]. Knowledge in an expert system using PAD-BASED is classified into two types: static knowledge, and dynamic knowledge. Static knowledge involves certain facts that are unchanged such as knowledge represented in the rule-based system. Dynamic knowledge changes from one situation to another situation such as knowledge obtained from the computational procedures. Knowledge representation in the PAD-BASED system can be any technique such as rule-based, semantic networks, frames, shell, external procedures, or a hybrid system. Figure 3 displays the PAD-BASED technique used as a knowledge organization in a PAD-BASED expert system.

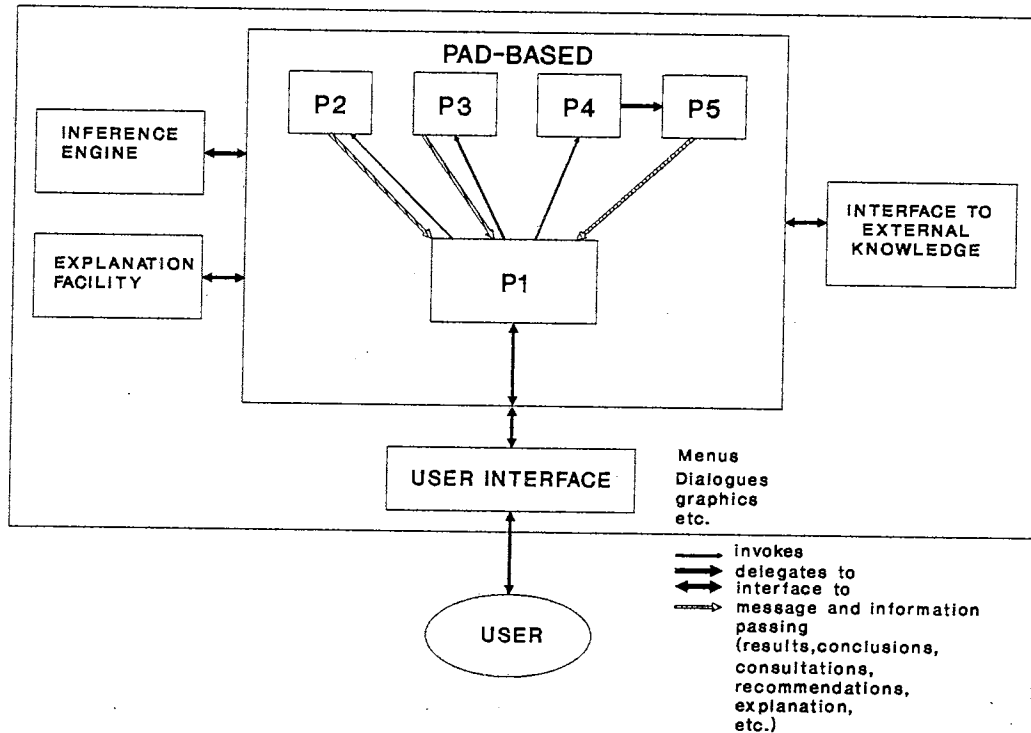


Figure 3. A PAD-BASED Expert System Architecture

In PAD-BASED system, the expert system consists of five basic components:

- 1) a knowledge database (facts and problem-solving methodology) organized in the PAD-BASED structure,
- 2) an inference engine,
- 3) an explanation facility,
- 4) an interface to external knowledge, and
- 5) a user interface.

Knowledge database in the PAD-BASED system is defined as a collection of prototypes. The interactions among prototypes are generated via delegation mechanism, and simple message passing. Delegation is a mechanism that allows prototype to delegate responsibility to another prototype to perform a task, or operation for the delegator. A

simple message passing is a mechanism that provides information transferring among prototypes. Invoking is a mechanism that initiates an operation of a prototype. An inference engine is a control structure in an expert system that is used to perform the inference reasoning tasks or to make deductions. The inference engines widely used in expert system are such as forward chaining and backward chaining. Furthermore, an inference engine also supports various knowledge acquisition, explanation facility, and user interface subsystems. An explanation facility provides access for a user to query the expert system on why and how the particular recommendation is derived. An interface to external knowledge is used to provide communication (retrieve or store) for the expert system and external knowledge such as databases, external computing modules, and others. Finally, a user interface is a component that a user communicates with the expert system by giving facts and queries to the system, and receiving result, recommendation, and explanation from it.

#### **4.2.1 Prototype Basic Elements**

Knowledge database in the PAD-BASED system consists of a collection of prototypes. Each prototype is comprised of four basic parts: name, knowledge, basic behaviors, and interfaces as shown in Figure 4. Name is a unique identification for the prototype such as "TRACTOR". A prototype contains a piece of knowledge describing the concepts or the characteristics of the prototype. For instance, Horse power=50, Begining age=NEW, and Purchasing price=\$20,000 are the knowledge describing a tractor. Basic behaviors are general behaviors relevant to the knowledge of the prototype. Typically, behaviors in a prototype are basically the same as behaviors of any expert

system, i.e., to accomplish a goal or subgoal, to give diagnosis, to provide recommendation or consultation, or to evaluate new situation. For instance, the basic behaviors of TRACTOR are "TRACTOR INFO", and "FACT ABOUT TRACTOR". "TRACTOR INFO" contains a method to retrieve a tractor list from an external file and present it to the user who in turn provides the facts to the prototype. This feature makes the prototype act slightly different from the conventional prototype since it contains another part called interfaces. Via its interfaces, a prototype must be able to:

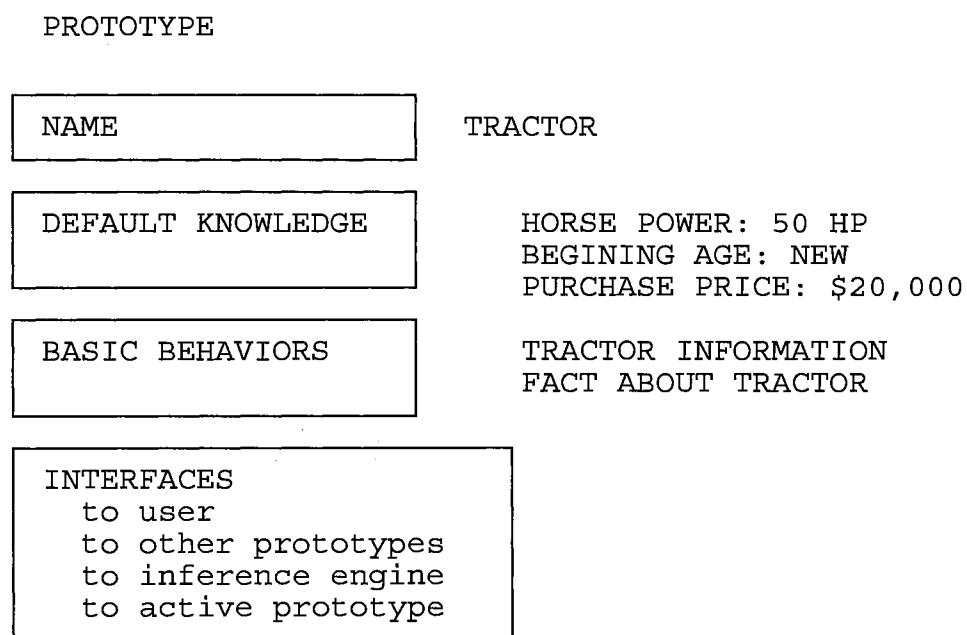


Figure 4. Basic Elements of a Prototype

- communicate with the user through the user interface such as using menus, dialogues, graphics, or others;
- access a reasoning technique such as using an inference engine to make inferences;
- delegate to other prototypes that share knowledge to this prototype, i.e.,

- contains a link to the other prototypes; and
- provide a simple message passing from the delegatee back to the object who first sends the message.

#### **4.2.2 Interaction among Prototypes**

In this section, an example is given to illustrate the interaction among prototypes in the PAD-BASED system. Four prototypes are used. They are MACHINERY, HARVESTING, TRACTOR, and IMPLEMENT. The structure of these prototypes is described in Figure 5. To simplify the illustration, let the user invoke the MACHINERY prototype. The user would like to obtain some knowledge about machinery for a harvesting operation. The following messages are sent to ask the HARVESTING prototype.

- 1) What is your operation?
- 2) What is the information about your equipment?
- 3) What is the minimum horse power recommended for your equipment?

The HARVESTING prototype looks up in its default knowledge and answers the first question that its operation is "Cutting". However, it cannot answer the second question, thus it delegates the message to the IMPLEMENT prototype. The delegatee (IMPLEMENT) obtains the knowledge about operation type (Cutting), equipment name (Pt-swather) from the delegator (HARVESTING), and the original message sender (MACHINERY). IMPLEMENT then invokes its method called "IMPLEMENT INFO" to infer the knowledge about implement equipment based on the cutting operation using pt-swather equipment. This knowledge is sent back to MACHINERY via message passing.

```

= Prototype A =
Name: Tractor
Default knowledge
  Horse Power: 50 HP
  Purchasing Price: $20,000
  Tractor Age: NEW
Basic behaviors
  Tractor info
  Facts about Tractor

```

```

= Prototype B =
Name : Implement
Default Knowledge
  Operation Type : Baling
  Equipment Name : Large Square Baler
  Purchasing Price : $54,000
  Age : NEW
  Size: 30.00 Ft
  Speed : 7 MPH
  Efficiency :
  Capacity :
Basic Behaviors
  Implement Info

```

```

= Prototype C =
Name : Harvesting
Default Knowledge
  Operation : Cutting
  Field acre : 80 acre
  Number of Cuttings : 4
  Cutting Equipment : Pt-swather
Basic Behaviors
  Harvesting Costs

```

```

= Prototype D =
Name : Machinery
Default Knowledge
  Tractor Horse Power : 125
  Tractor Age : 3 years
  Price : $46,000
Basic Behaviors
  Machinery info

```

Figure 5. Example of Prototypes in PAD-BASED System

HARVESTING has one more question to answer. This message is delegated to the TRACTOR prototype to invoke the behavior called "FACTS ABOUT TRACTOR". This behavior requires only the equipment name from HARVESTING to look up in its knowledge about minimum horse power for that particular equipment. The answer is sent back to MACHINERY prototype. Figure 6 illustrates the delegation mechanism and message passing for the given example.

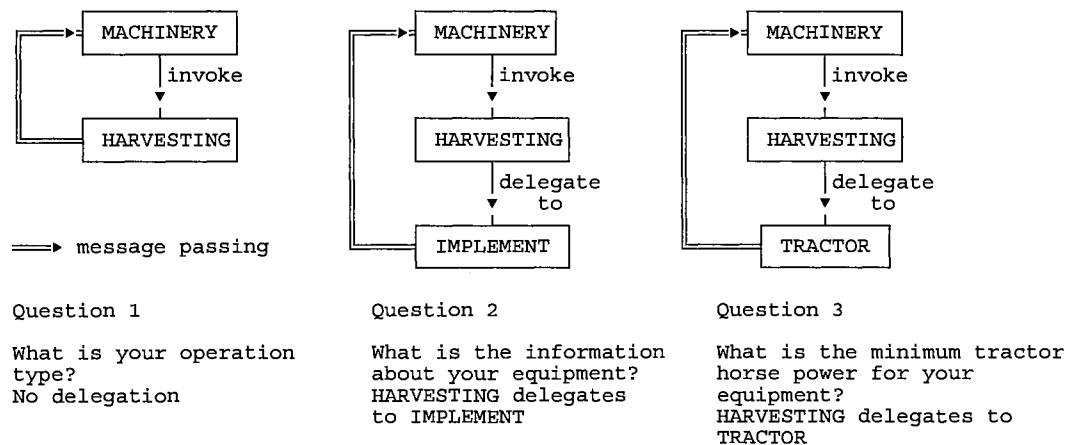


Figure 6. Delegation Mechanism and Message Passing



## **CHAPTER V**

### **IMPLEMENTATION OF THE PAD-BASED MODEL**

The PAD-BASED model is implemented in an agricultural application of an expert system for weed management. The following sections describe the nature of agricultural applications and introduce the PAD-BASED development life cycle model to design and develop an expert system called WEEDPLUS which will be employed by Oklahoma farmers. The KnowledgePro® by Knowledge Garden Inc. is used as a development tool. This implementation highlights three important advantages of the PAD-BASED model: minimizing the development time for an expert system, increasing knowledge sharing and re-use among the structured expert systems, and providing an alternate memory management scheme to develop an expert system in a small system with limited internal memory.

#### **5.1 Agricultural Applications**

Agriculture is an important area for human life since it serves the basic needs of food. Proper integrated pest management practices are mandatory for the safety of human life based on food supply. Because this area contains several domains of expertise (Agricultural Engineering, Agricultural Economics, Entomology, Agronomy, Soil and Fertility Science), the main problem in this area is how to incorporate and deliver

knowledge from these experts to farmers and agribusinesses. These common problems in agricultural applications such as crop and livestock production involve a decision support system, a scheduling system, and an integrated pest management system. As stated by Naegele and others [Naegele et al. 89, Coulson et al. 89], the concept of integrated pest management (IPM) is essential and has been developed to the point that it is advanced and very well established among the experts in the area. This IPM fundamental concept is composed of analytical models, management models, technical information, and expert recommendations which are entirely too complicated to be investigated by the farmers. As a result, the critical issue appears to be how to use efficiently and effectively this knowledge to solve the real problem. In fact, the major problem is how to integrate, implement, and deliver this knowledge for problem solving and decision making.

Based on the complex nature of the problem and the knowledge interactions among the experts, expert system has been chosen to solve this problem. Because an expert system contains the facility to organize the knowledge such that it is able to draw a conclusion and an explanation on the result, the expert system technique has been successfully used in the agricultural area. Furthermore, expert systems provide an interactive environment to deliver knowledge to the users who mostly have the limited knowledge about computer. Knowledge acquisition technique provides a means for the knowledge engineer to acquire and capture knowledge from the experts, while knowledge representations such as rule-based, semantic network, frames, and shell systems are used to organize and represent the knowledge. In the last decade, hundreds of expert systems have been developed for many crops such as cotton, wheat, barley, corn, soybean, and

alfalfa. The first goal for developing these expert systems is to provide the farmers with sound management advice, especially in an area that is hard to seek advice from an expert directly. The second goal is to use the expert system as an educational tool by the extension people to give advice to the farmers.

Developing the expert systems for farming management and decision support has been very successful. However, there are three significant issues left out. First, developing an expert system is time consuming. For instance, PROFALF [Limsupavanich et al. 92], HAYMACH\$ [Huhnke et al. 90], NPK\$PLUS [Johnson and Nofziger 90], and WEEDALF [Stark et al. 89] are expert systems being used by the alfalfa growers providing information about alfalfa pest management and alfalfa economics. Each system required years of design and development, because of the interactions among several disciplines. Second, the concepts of knowledge sharing and re-use are ignored. For example, two expert systems are developed for the same crop (alfalfa) in different states. PROFALF expert system has been developed at Oklahoma State University while Alfalfa Management Expert System has been developed at Purdue University [Rhykerd et al. 92]. Both expert systems are designed to incorporate information on site, soil texture, drainage conditions, soil fertility, seed variety, pest control management, machinery and operation cost. If the concepts of knowledge sharing and re-use had been developed, it would optimize the usage of expert systems while minimizing the effort and the amount of development time. As in the above example, instead of building two different expert systems, the common knowledge such as pest control management, machinery information, and operation cost can be shared and re-used both in Oklahoma and Indiana. The different knowledge such as location and weed types can be added as a separate

module or prototype. Consequently, years of the effort on development time can be minimized and the common knowledge can be shared and re-used. Furthermore, knowledge sharing and re-use concept not only applies to a particular crop but also applies to the system as a whole. For example, considering a crop production system, each crop consists of a similar system such as planting (seedbed preparation, seeding process, and fertilization), pest management practices (insecticide and herbicide applications), harvesting, and marketing. Knowledge about insecticide, herbicide, and machinery information is common. This knowledge can be organized into separate modules for sharing and re-use. Then the different knowledge is defined for a specific crop or a particular location.

Finally, expert systems usage among the farmers is limited because of the small PCs that they own could not run most of the applications delivered. To increase the expert systems usage among the farmers implies that an expert system needs to be developed for this limited internal memory environment. Although the computer technology has changed tremendously in this decade, there are still a number of users who are struggling due to the memory limitation of small PCs such as computer systems based on the Intel 286 CPU microchip. Among these users are the farmers who face this limitation. Thus, one of the basic problems in the expert system area is dealing with how to design and implement the expert system to run on small PCs. Particularly, how to implement WEEDPLUS in a small PC is yet another crucial issue in this chapter.

In the following sections, the development process of a PAD-BASED expert system is illustrated. The advantages of the PAD-BASED model toward the cited problems are described in the next chapter.

## 5.2 PAD-BASED Expert System Development Life Cycle

Software life cycle is a term generally used as the process of developing and maintaining software. In conventional software development life cycle, waterfall model [Royce 70, Boehm 81] is one of the popular phased model being used in software engineering. This model consists of five phases: Analysis, Design, Implementation, System Testing, and Maintenance. Rapid prototyping approach, a prototyping model, is another model used for fast implementation of a part of the system to demonstrate its functionality, and to meet user and manager requirements.

In expert system development, knowledge engineering is a term describing the process of developing and maintaining expert systems [Turban 92]. Knowledge engineering involves the cooperation of human experts and knowledge engineers to explicitly extract and implement the knowledge and methods that the human experts use to solve real problems. Many of these problems are termed *knowledge-intensive problems* because their data and problem-solving methods are not explicit knowledge [McGraw and Harbison-Briggs 89].

In general, a typical conceptual structure of an expert system development is viewed in figure 7 [McGraw and Harbison-Briggs 89]. Its development process consists of the knowledge engineer acquiring knowledge from the human experts while gathering more information from the documented knowledge such as book. The knowledge engineer then selects the building tools and languages to be used in implementation. Finally, the expert system is developed and tested. This expert system development life cycle is similar to the conventional software development life cycle described above. However, knowledge acquisition from human experts is more complicated and usually

needs extensive time. In addition, building and testing expert systems are lengthened because of their complex interaction and unstructured nature.

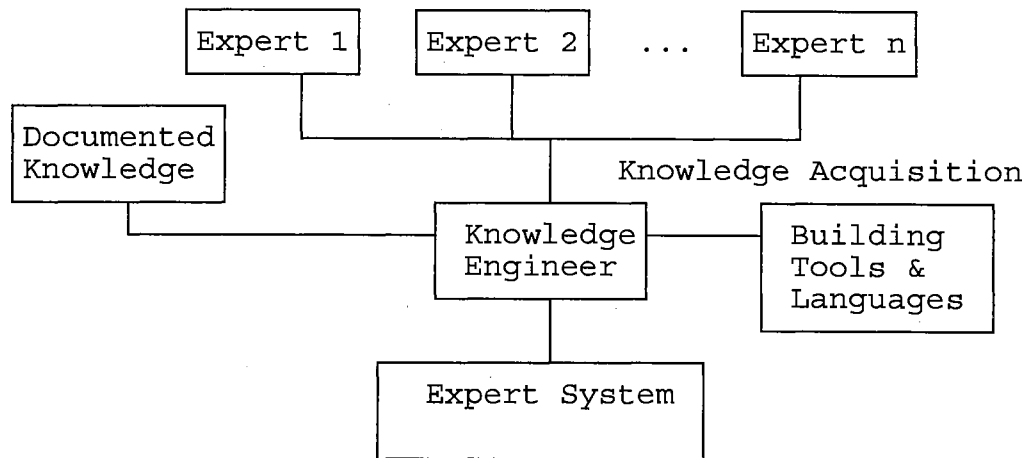


Figure 7. A Simple Conceptual Structure for Expert System Development  
(Source: Adapted from E. Turban, Macmillan, New York, NY, 1992, p. 120)

Rapid prototyping approach has been widely used in developing expert systems [Redin 87], [Cholawski 88]. A rapid prototyping entails the selection of knowledge representation tools and techniques to perform rapid development of a section of the expert system, testing on the initial system, iterative refinement, and further development [McGraw and Harbison-Briggs 89]. In particular, rapid prototyping is used for quick development of an initial version of a small part of the expert system to test the effectiveness of the knowledge representation and inference mechanisms to solve a particular problem. Then the prototype system is refined or modified several times until the system is acceptable or meets the requirement. As stated by McGraw and Harbison-Briggs, rapid prototyping can be used successfully in developing solutions to knowledge-intensive problems if the problem is sufficiently small, does not require maintenance or

modification. In addition, a tool should be available for developing the prototype.

Figure 8 illustrates the application of the PAD-BASED model to the expert system development structure. The experts and knowledge engineers agree on the task organization [Chandrasekaran et al. 92], i.e., tasks/subtasks division (T11, T12, ..., T1n; T21, T22, ..., T2n; ...; Tn1, Tn2, ..., Tnn) of the application. The knowledge engineer then uses any approach to perform knowledge acquisition on each subtask, for instance, *the model approach* (using the existing model that is well-suited to that particular domain to develop a set of facts and rules) or *the team approach* (the domain experts and knowledge engineers work closely together for an extended period of time and develop a model and computer program for that problem) [McGraw and Harbison-Briggs 89]. Consequently, the task organization reduces the size of the problem into several small subtasks which directly reduces the time and effort in knowledge acquisition on each subtask. In addition, the prototype structure (see figure 4) is recommended to be used as the knowledge acquisition method on each subtask so that knowledge on each subtask will be very well organized and highly structured.

Then each of the knowledge engineers constructs the prototypes (P11, P12, ..., P1n; P21, P22, ..., P2n; ..., Pn1, Pn2, ..., Pnn) for the subtasks with complete information. In this stage, the strategy adopted from Walter and Nielson [Walter and Nielson 88] is used to build a separate prototype for each subtask and the integration is performed in the final stage. Prototype testing is executed via delegation mechanism and message passing.

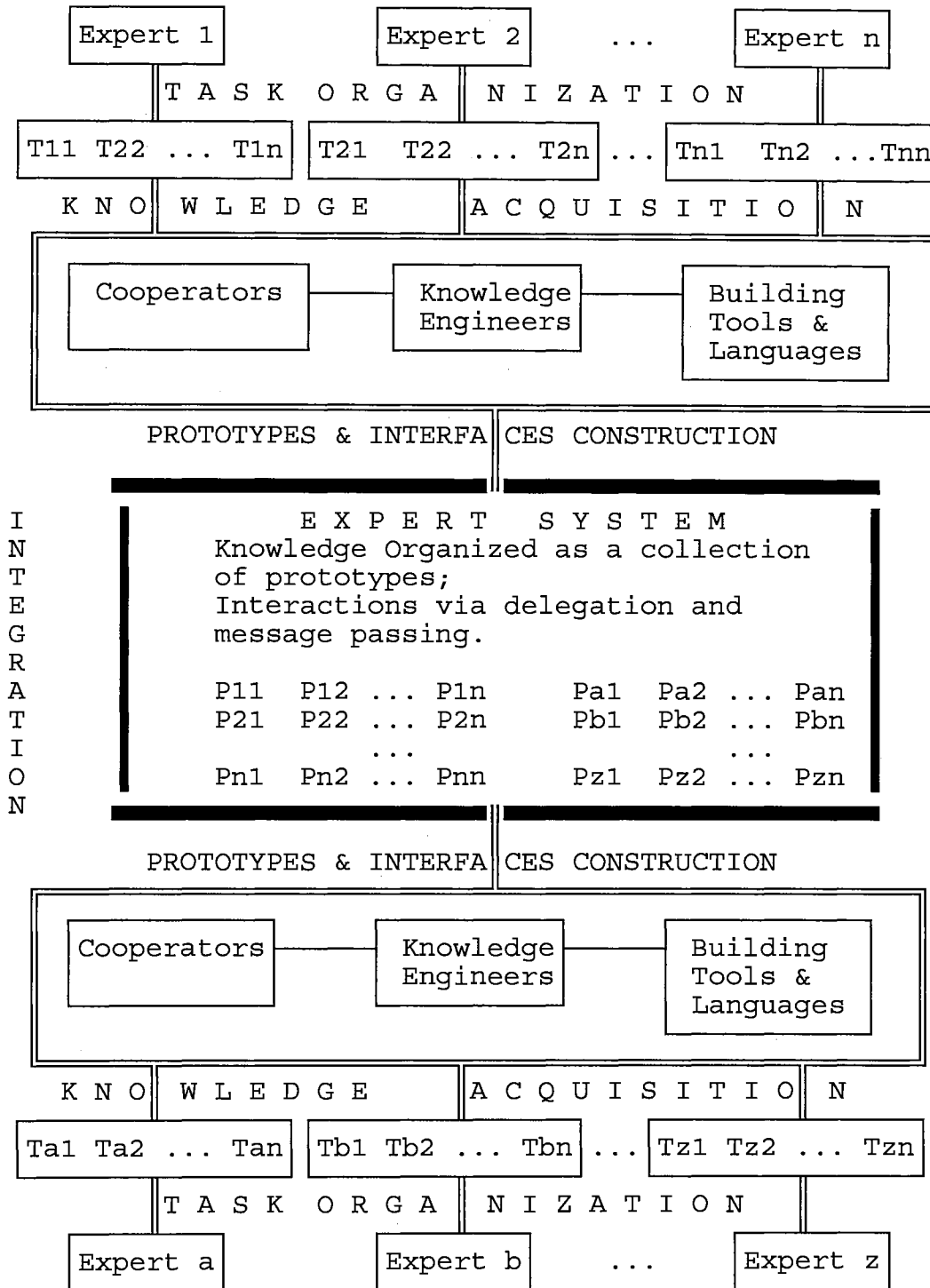


Figure 8. The PAD-BASED Expert System Development Structure;  
 Tij represents a Task;  
 Pij represents a Prototype



Because each prototype contains complete knowledge of a subtask together with its interface to other prototypes, knowledge acquisition, knowledge representation, prototype development, and testing can be performed concurrently. Thus, both the experts and the knowledge engineers can work in parallel. Finally, the integration of the prototypes is performed. This development process is fully described below.

From the above demonstration, the PAD-BASED expert system development life-cycle model is presented to develop a PAD-BASED expert system. This development life-cycle model consists of five phases: System Requirements, Conceptual System Design and Analysis, System Development, System Integration and Evaluation, and System Maintenance. Figure 9 exhibits this model.

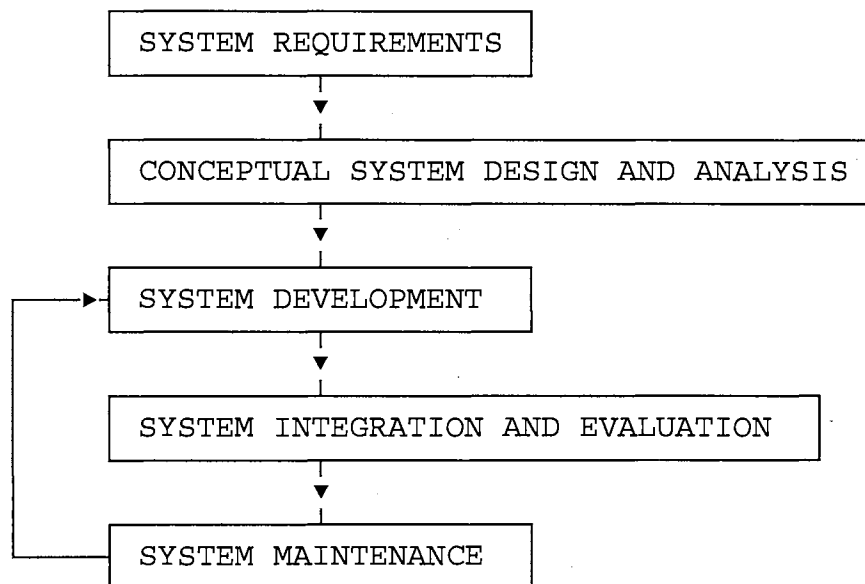


Figure 9. The PAD-BASED Expert System Development Life-Cycle

*System Requirements* explore the general view of the problem, problem-solving strategies, the selected expert system techniques and tools, and the analysis of risks and benefits before developing the expert system. *Conceptual System Design and Analysis* involves detailed study of the problem and its scope, problem-solving methodology, system interface and structure. In this phase, the conceptual design of the system is completed. Techniques such as flow diagram or pseudocode can be used to present the design of the system. Also, the analysis of the system complexity, the appropriation of selected techniques and tools, and the cost/benefit of the development are seriously evaluated. In addition, the development time frame is proposed as well.

*System development* is the main phase in which the PAD-BASED structure is particularly involved. This phase is comprised of: Task Organization, Task Knowledge Acquisition, Prototypes and Interfaces Construction, and Prototype Testing. *Task Organization* includes task division and task identification. In this step, the problem is divided into small subtasks ( $T_{ij}$  in figure 8). The scope of each subtask, knowledge, problem-solving technique, and the interaction to other subtasks are all defined in this step. The prototype structure (see figure 4) is used as a means for knowledge acquisition on each subtask. Consequently, the default knowledge and basic operations of a subtask are acquired, and the interactions or interfaces to other subtasks are drawn. *Prototypes and interfaces construction* deals with the system implementation. The selected development tool and knowledge representation techniques are used for prototype construction corresponding to the subtask division and identification in the previous step ( $T_{ij} \Rightarrow P_{ij}$  in figure 8). The interface section of each prototype is constructed to provide the path for delegation and message passing mechanisms. For instance, prototype

P1 (delegator) delegates to prototype P2 (delegatee) to use the method m2. Interface section of P1 is constructed to provide a delegation path to P2 while sending all the necessary information to P2 to activate method m2. Then the result is passed back to P1 which requires this shared operation. *Prototype Testing* can be done after the complete construction of each prototype and interface except if it required shared knowledge/operations from the prototype that has yet to be constructed. This restriction is generally minimized by building the prototype (delegatee) that contains sharing knowledge/operations before building the prototype (delegator) that requires shared knowledge. Thus, system development in this model provides a parallel or a concurrent environment between the task knowledge acquisition, prototypes and interfaces construction, and prototype testing as demonstrated in figure 10.

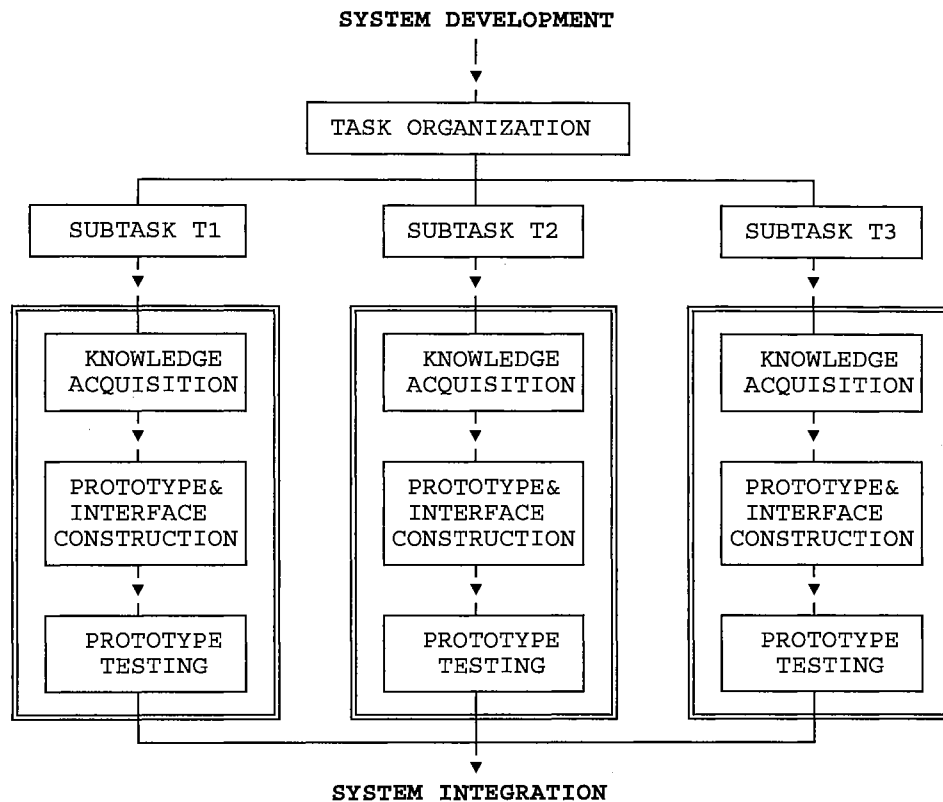


Figure 10. System Development Phase in the PAD-BASED Development Life Cycle

*System integration and evaluation* focuses on the interfaces between the prototypes and the interfaces to the external module (if required). The emphasis here deals with the interfaces among prototypes. These interfaces are defined via delegation and message passing mechanisms. All the paths providing for these mechanisms are previously defined in the prototypes and interfaces construction. Thus, system integration among prototypes is minimal. System evaluation is the testing stage of the whole system to determine the degree of accuracy and degree of usage which directly involves satisfaction of both the experts and the users. *System maintenance* includes system modification and system augmentation. This phase is comparable to the maintenance phase in the waterfall model. System modification and augmentation are primarily done repeatedly from the system development phase. Each prototype can be modified without interfering with the function of other prototypes. Also, adding a prototype to the system can be done easily by constructing the prototype and the paths to access this prototype and to the other prototypes to which it needs to delegate.

The PAD-BASED development life cycle model provides three advantages to the rapid prototyping model. First, it reduces the iterative refinement step which implies decreasing an amount of development time. Second, it produces the actual expert system that is ready for delivery while rapid prototyping produces the operational prototype and a rather small system. Finally, it produces a highly structured system which simplifies system maintainability as described above.

### **5.3 WEEDPLUS: Weed Management Expert System**

In this section, the PAD-BASED model and the PAD-BASED development process are used to design and develop an expert system called WEEDPLUS [Mitrpanont et al. 93] which will be delivered to Oklahoma farmers for weed management. Weeds can be very harmful to crop production. If not properly controlled, weeds can reduce more than half of the crop production. In general, a weed scientist uses his expertise incorporated with field history and status, soil fertility information, crop status, weed status, weed management, and herbicide information to determine weed problem, and to estimate yield loss based on the management practice of the farmers. Because there are thousands of weeds and each of them has its own characteristics, its competitiveness to different crops and to field status, knowledge about weed management is usually developed as a part of a specific crop management system. For example, WEEDALF [Stark et al. 89] is designed for a weed management system for alfalfa established stands and contains only a subset of weeds and herbicides. To extend WEEDALF for use with weed management in alfalfa seedling (preplanted) stands is complicated because of its non-structured knowledge organization. Moreover, to share or re-use knowledge such as herbicides in WEEDALF to the other crops needs even more work.

### **5.4 Designing WEEDPLUS Expert System**

To design WEEDPLUS expert system, three main steps are concerned: task organization, prototypes and interfaces construction, and system integration (i.e., interactions among prototypes in WEEDPLUS). Based on the PAD-BASED development process, a problem is divided into small subtasks. Each subtask consists of:

- knowledge about the task,
- basic behaviors/operations such as problem-solving methods, and
- its interactions to other subtasks.

Knowledge acquisition is performed according to each subtask. After a knowledge engineer obtains a complete detail of that subtask, a prototype is developed and tested while knowledge for another subtask can be acquired.

#### 5.4.1 Task Organization

WEEDPLUS system is divided into three main tasks: weed management, weed information, and herbicide information. Figure 11 demonstrates the organization of the main tasks in WEEDPLUS.

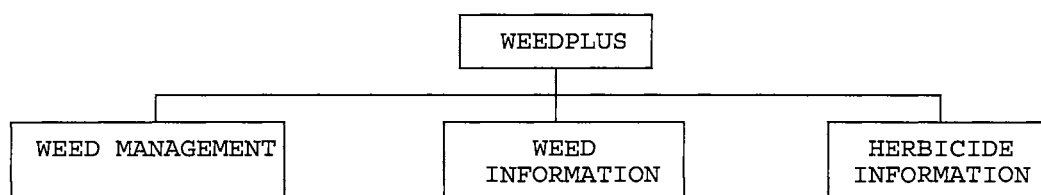


Figure 11. Organization of the Main Tasks in WEEDPLUS.

*Weed management* contains knowledge about weed management for alfalfa. It is divided into weed management for alfalfa seedling stands, weed management for alfalfa established stands, weed problems in both seedling and established stands, and economics or cost/benefit of the herbicide control. *Weed information* contains knowledge about weeds such as type, characteristics, and control. For simplicity, weed information contains only Oklahoma weeds. Similarly, *herbicide information* contains knowledge on

all herbicides that effectively control alfalfa weeds, their usage, their use rate and their application cost. Weed management for alfalfa subsystems can delegate to weed information and herbicide information to retrieve and access information on a specific weed and herbicide. Figure 12 elaborates this task/subtasks hierarchy.

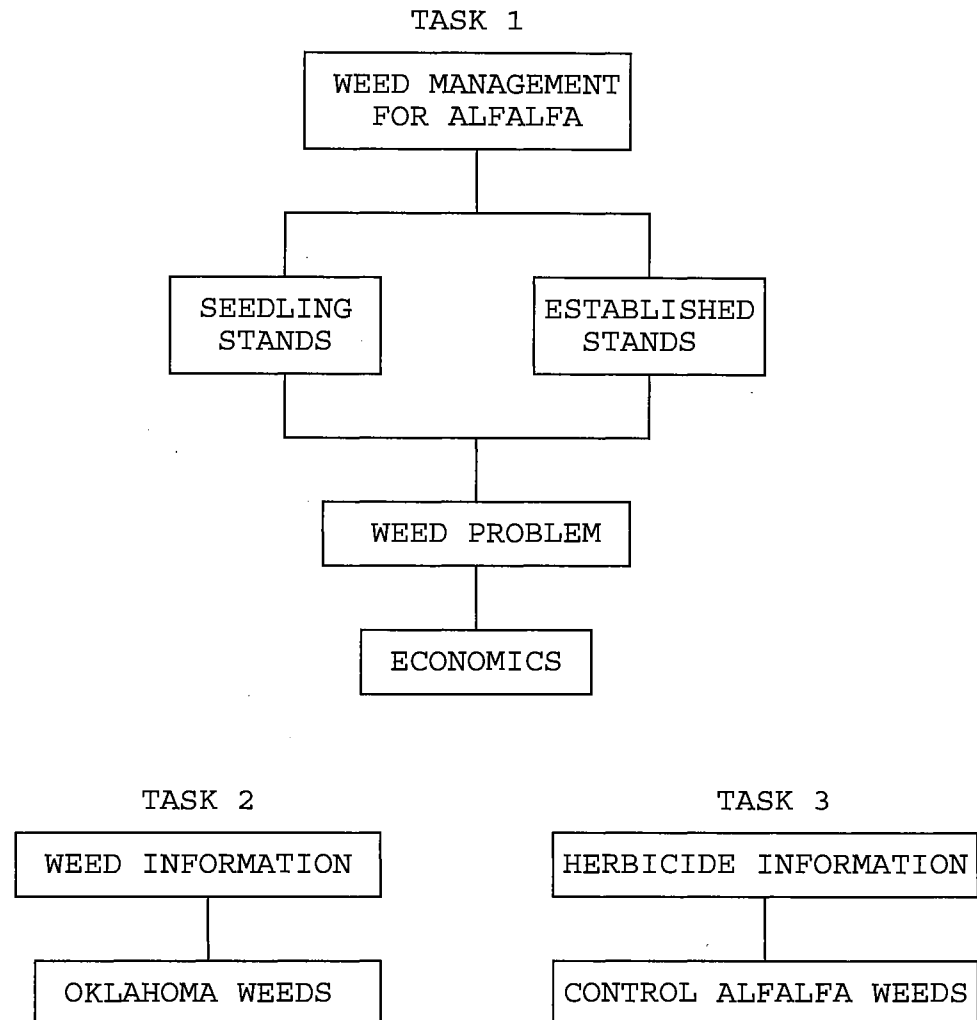


Figure 12. Subtasks Organization in Weed Management, Weed Information, and Herbicide Information

### 5.4.2 Prototypes and Interfaces Construction

Prototypes and interfaces construction is the next step in developing a PAD-BASED expert system. The knowledge engineer organizes knowledge in each prototype according to the task/subtask organization. Figure 13 illustrates the prototype construction corresponding to the task/subtask organization in figure 12. MAIN\_PROTOTYPE, WM\_PROTOTYPE (Weed Management), SL\_PROTOTYPE (Seedling Stands), ES\_PROTOTYPE (Established Stands), WP\_PROTOTYPE (Weed Problem), EC\_PROTOTYPE (Economics), WI\_PROTOTYPE (Weed Information), and HI\_PROTOTYPE (Herbicide Information) are the basic prototypes in WEEDPLUS.

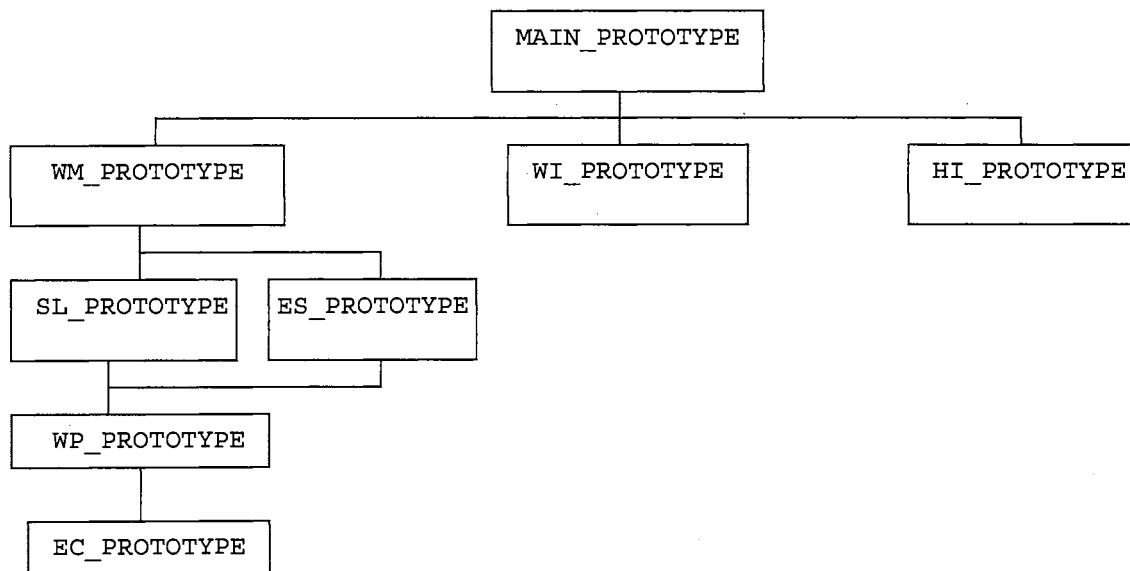


Figure 13. Prototypes Constructure Corresponding to Task Organization

In this example, the MAIN\_PROTOTYPE is developed to provide the main interface to switch the task among the WM\_PROTOTYPE, WI\_PROTOTYPE, and HI\_PROTOTYPE. MAIN\_PROTOTYPE is considered to be the original delegator for



each task. In addition, in small system knowledge and methods of MAIN\_PROTOTYPE should be minimized because MAIN\_PROTOTYPE will be active in the main memory as an original delegator. However, its interface section always plays the most important role in the application.

Figure 14 shows the basic components of some prototypes in WEEDPLUS. As an example, a prototype containing knowledge about weed management for alfalfa (WM\_PROTOTYPE) consists of knowledge about the task (STAND\_INFO, SEEDLING\_GEN\_INFO, ESTABLISH\_GEN\_INFO), and the basic operation (FIND\_RECM). STAND\_INFO contains knowledge about alfalfa stand status such as seedling or established stands. SEEDLING\_GEN\_INFO contains general information of the seedling stands such as planting time, previous crop, and previous weed problem. ESTABLISH\_GEN\_INFO contains general information of established stands such as current weed problem (perennial broadleaf weed, perennial grass, or dodder) and previous weed problem. FIND\_RECM utilizes the knowledge about stand information and previous weed problem in the field to determine the group of rules that can be used to provide the general recommendation.

DELEGATE\_TO\_SEEDLING and DELEGATE\_TO\_ESTABLISH are the interfaces that provide the paths for delegation mechanism to either SL\_PROTOTYPE (seedling) or ES\_PROTOTYPE (established). If the stand status is seedling stand then WM\_PROTOTYPE delegates to SL\_PROTOTYPE, otherwise to ES\_PROTOTYPE.

NAME:	WM_PROTOTYPE
KNOWLEDGE:	STAND_INFO SEEDLING_GEN_INFO ESTABLISH_GEN_INFO
METHOD:	FIND_REC_M
INTERFACE:	DELEGATE_TO_SEEDLING DELEGATE_TO_ESTABLISH

NAME:	SL_PROTOTYPE
KNOWLEDGE:	FIELD_INFO
METHOD:	DISPLAY_FIELD_INFO FIND_REC_M FIND_SP_REC_M FIND_FA_REC_M
INTERFACE:	DELEGATE_TO_WEEDPROBLEM

NAME:	WP_PROTOTYPE
KNOWLEDGE:	CS_IN_FALL
METHOD:	HERB_CHOICE H_OPTION PPI_OPTION POST_OPTION
INTERFACE:	DELEGATE_TO_HERBUNIT DELEGATE_TO_ECONOMIC

NAME:	HI_PROTOTYPE
KNOWLEDGE:	HERB_INFO
METHOD:	HERB_LIST HERB_SCREEN FIND_HERBFILE HERBCOST MIX_HERBICIDE
INTERFACE:	HERBFILE

NAME:	EC_PROTOTYPE
KNOWLEDGE:	ECONOMIC_INFO
METHOD:	SEEDLING_ECONOMIC ESTABLISH_ECONOMIC WHAT_NEXT FIND_EST_YIELD

Figure 14. Examples of Prototypes in WEEDPLUS.

## 5.5 Interactions among Prototypes in WEEDPLUS

In this section, an example of the interactions among prototypes in WEEDPLUS is illustrated. To simplify the illustration, WEEDPLUS is consulted on weed management in alfalfa seedling stands. The following provides the consultation process:

- 1) WEEDPLUS queries the user about the basic information of the alfalfa stands.
- 2) WEEDPLUS gets more specific information of seedling stands and give basic recommendations.
- 3) WEEDPLUS queries on the specific weed problem and provides the proper herbicide option.
- 4) WEEDPLUS presents information on the selected herbicide and determines the cost-benefit on weed control.

WEEDPLUS starts the process by invoking MAIN\_PROTOTYPE which requests basic information from the user so it can delegate to the proper prototype. For alfalfa weed management, MAIN\_PROTOTYPE delegates to WM\_PROTOTYPE which queries the general information for alfalfa stands and determines the fundamental recommendation based on the specific situation. WM\_PROTOTYPE delegates to SL\_PROTOTYPE to obtain the information of the seedling stands and the recommendations for the alfalfa seedling stands by using FIND\_RECM method. SL\_PROTOTYPE delegates to WP\_PROTOTYPE to perform CS\_IN\_FALL method to consult on a fall-planted cool-season weed problems and also to provide the herbicide option for that particular weed problem. WP\_PROTOTYPE delegates to HI\_PROTOTYPE to use FIND\_HERBFILE method to obtain the information of the specified herbicide. Finally, WP\_PROTOTYPE

delegates to EC\_PROTOTYPE to use SEEDLING\_ECONOMIC method to determine the cost-benefit of the weed control.

In the above interactions, MAIN\_PROTOTYPE is the original delegator that needed the consultation. Therefore, MAIN\_PROTOTYPE receives all information from the delegatee that performed the operation for WM\_PROTOTYPE. Figure 15 demonstrates part of the delegation mechanism of the above consultation.

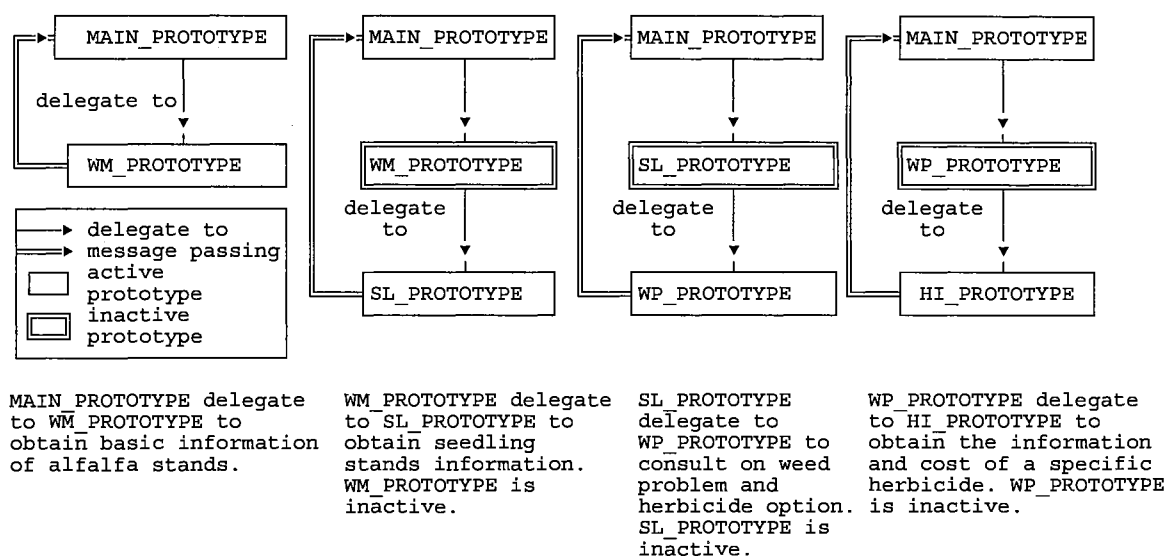


Figure 15. Delegation Mechanism in WEEDPLUS.

## **CHAPTER VI**

### **ADVANTAGES OF THE PAD-BASED MODEL**

In this chapter, the advantages of the PAD-BASED model are presented. The emphasis is on three main issues: to reduce development time, to provide an alternate memory management scheme in small computer system, and to increase knowledge sharing and re-use internally and externally. Internal knowledge sharing and re-use means knowledge sharing and re-use in a PAD-BASED expert system, while external knowledge sharing and re-use implies knowledge sharing and re-use between a PAD-BASED expert system and a structured existing expert system.

#### **6.1 Reducing Development Time**

In general, the development process of an expert system can be lengthy [Turban 92]. In particular, developing a large expert system involving many human experts in several areas requires years of design and development. Although, the expert system development life cycle is similar to the conventional software development life cycle, knowledge acquisition from human experts is complex and needs excessive time. In addition, expert system developing, integration, and testing processes are lengthened by its complex interactions and unstructured nature.

In order to reduce the expert system development time, the modularity and structurability of the expert system must be increased in each level such as the knowledge

acquisition, system design, system development, knowledge representation, system testing, and system integration. In section 5.2, the PAD-BASED expert system development life cycle model has been described. The system development phase (see figure 10) is fully elaborated in such a way that it clearly elucidates the concurrency of task knowledge acquisition, knowledge organization (i.e., prototypes and interfaces construction), and prototype testing. This process directly helps reduce the development time. First, knowledge acquisition, knowledge representation, system development, and system testing are performed parallelly not sequentially. Second, the knowledge engineers and the experts have the same clear picture of the whole system from the task/subtask organization (in the large scale) and the knowledge organization (in the small scale, i.e., the prototype structure) which helps keep them in the same consistent format. Third, since the prototype structure is used as a means to knowledge acquisition on each subtask, the structurability and modularity concepts have been introduced into the early phases such as the system design. The implication is that the PAD-BASED structure can be used to increase the structurability in expert system design. Fourth, using the prototype structure and the delegation mechanism simplifies expert system testing and system integration. Because each prototype in the PAD-BASED expert system typically contains knowledge about subtask, its methods/operations, and interfaces or the paths for its interactions to other subtasks, this particular structure makes the prototype a self-defined unit. Testing can be done in a miniature scale; i.e, in each prototype which implies more accuracy in the integration stage. Besides, system integration is performed in a more direct fashion because most of the delegation paths which provide the interactions among the prototypes have been defined at the prototypes and interfaces construction step. This construction

directly simplifies system integration. Finally, maintaining a PAD-BASED expert system is simplified because of its highly structured organization.

## **6.2 Providing an Alternate Memory Management Scheme for Small Computer System**

Although computer technology has changed tremendously in this decade, there are still a number of users who are struggling due to the memory limitation of small PCs such as the 286 computer systems. Specifically, designing an expert system to run on the system with limited internal memory is one of the critical issues in delivering expert systems to their general users who mostly own small PCs.

A design objective of WEEDPLUS is to run on small systems with limited internal memory. Therefore memory usage is critical issue in this expert system development. Delegation mechanism is the key approach to memory management in WEEDPLUS to run in the 286 system. The interface section of the delegator plays the most important role in memory management since it provides the paths or links to the delegates which contain the shared knowledge/operations. The main idea of this memory management scheme is to keep the original delegator which requires the shared knowledge active in the memory. The delegatee prototype is loaded into the main memory to perform the requested services and is removed after it has completed the operation. Figure 16 shows the memory organization in WEEDPLUS.

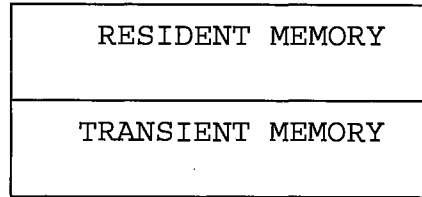


Figure 16. Memory Organization in WEEDPLUS

Memory in WEEDPLUS is divided into two parts: resident memory and transient memory. *Resident memory* basically contains the delegator prototype, its knowledge, methods and its interfaces. In addition, it also contains all the knowledge derived or obtained from the other prototypes either by the invoking or delegation mechanism. *Transient memory* is used as a temporary or working space for the delegatee prototype to perform the specific methods for the delegator. After it has completed the requested operations, it is then removed out of the transient memory. The new knowledge obtained from the delegatee is stored in the resident memory for other usage. The following illustrates this memory management scheme during this consultation in WEEDPLUS:

- 1) a user invokes WEEDPLUS.
- 2) WEEDPLUS queries the user about the basic information of the alfalfa stands.
- 3) WEEDPLUS gets more specific information of seedling stands and gives basic recommendation.
- 4) WEEDPLUS queries on the specific weed problem and provides the proper herbicide option.



## STEP 1: MAIN\_PROTOTYPE IS ACTIVATED

==> MAIN\_PROTOTYPE is active

```
!MAIN
* MAIN_PROTOTYPE
...
  WEED MANAGEMENT FOR ALFALFA
  DELEGATE_TO_WEEDMGMT
```

## STEP 2: MAIN\_PROTOTYPE DELEGATES TO WM\_PROTOTYPE

==> MAIN\_PROTOTYPE is resident  
WM\_PROTOTYPE is transient

```
...
MAIN_PROTOTYPE
...
  DELEGATE_TO_WEEDMGMT
  MANAGEMENT
* WM_PROTOTYPE
...
  DELEGATE_TO_SEEDLING
  DELEGATE_TO_ESTABLISH
```

## STEP 3: WM\_PROTOTYPE DELEGATES TO SL\_PROTOTYPE

==> WM\_PROTOTYPE is removed  
MAIN\_PROTOTYPE is resident  
SL\_PROTOTYPE is transient

```
...
MAIN_PROTOTYPE
...
  DELEGATE_TO_WEEDMGMT
  MANAGEMENT
  DELEGATE_TO_SEEDLING
  SEEDLING
* SL_PROTOTYPE
...
  DELEGATE_TO_WEEDPROBLEM
```

## STEP 4: SL\_PROTOTYPE DELEGATES TO WP\_PROTOTYPE

```

====> SL_PROTOTYPE is removed
        MAIN_PROTOTYPE is resident
        WP_PROTOTYPE is transient

```

```

...
MAIN_PROTOTYPE
...
DELEGATE_TO_WEEDMGMT
MANAGEMENT
DELEGATE_TO_SEEDLING
SEEDLING
DELEGATE_TO_WEEDPROBLEM
* WP_PROTOTYPE
...
DELEGATE_TO_HERBICIDE
DELEGATE_TO_ECONOMICS

```

The partial code above shows that only the original delegator which required the operation is staying alive or active in resident memory (MAIN\_PROTOTYPE). Figure 17 shows the active and inactive prototypes during the above consultation. The delegatee that uses the transient memory as working space is removed after it has performed the requested operation. The new knowledge or result is passed to be stored in the resident memory as the global variables for later usage. If these global variables use too much space, they are stored in the information file which will be used later. This information file can be used as the knowledge passing or sharing vehicle to the other expert systems. Particularly in WEEDPLUS, the information file is intended to share knowledge to PROFALF expert system in the section of weed management for alfalfa. For the best memory usage especially in the 286 system, the knowledge engineer should work very carefully in both the task organization, and the prototypes and interfaces construction steps.

	TIME	ACTIVE PROTOTYPE	INACTIVE PROTOTYPE*
STEP1	t1	MAIN_PROTOTYPE	
STEP2	t2	MAIN_PROTOTYPE WM_PROTOTYPE	
	t3	MAIN_PROTOTYPE	WM_PROTOTYPE
STEP3	t4	MAIN_PROTOTYPE SL_PROTOTYPE	
	t5	MAIN_PROTOTYPE	SL_PROTOTYPE
STEP4	t6	MAIN_PROTOTYPE WP_PROTOTYPE	
	t7	MAIN_PROTOTYPE	WP_PROTOTYPE

\* is removed out of the main memory

Figure 17. Active and Inactive Prototypes during the Consultation

Delegation mechanism is implemented by using a simple technique of loading and removing the prototypes in and out of the memory. In KnowledgePro®, the command such as LOAD() is used to load the prototype into the memory and the command REMOVE\_TOPIC() is used to remove the prototype (or topic) and its variables out of the main memory. The following is a sample set of instructions of the *interface section* called DELEGATE\_TO\_SEEDLING in step 3 above which removes WM\_PROTOTYPE and delegates to SL\_PROTOTYPE. A partial code in the SL\_PROTOTYPE is also provided.

```

TOPIC DELEGATE_TO_SEEDLING.
  REMOVE_TOPIC(WM_PROTOTYPE).
  LOAD('SEEDLING.HKB').
  DO(SEEDLING).
  REMOVE_TOPIC(SEEDLING).
END.

```

```

=="SEEDLING.HKB"==

TOPIC SEEDLING.
  DO(SL_PROTOTYPE).
  REMOVE_TOPIC(SL_PROTOTYPE).
  DO(DELEGATE_TO_WEEDPROBLEM).
  REMOVE_TOPIC(DELEGATE_TO_WEEDPROBLEM).
END.

TOPIC SL_PROTOTYPE.
  DO(FIELD_INFO).
  DO(DISPLAY_FIELDINFO).
  DO(FINDRECM).
  TOPIC FIELD_INFO.
  ...
  END.
  TOPIC DISPLAY_FIELDINFO.
  ...
  END.
  TOPIC FINDRECM.
  ...
  END.
  TOPIC FIND_SP_REC.
  ...
  END.
  TOPIC FIND_FA_REC.
  ...
  END.
END. (* end sl_prototype *)

TOPIC DELEGATE_TO_WEEDPROBLEM.
  REMOVE_TOPIC(SL_PROTOTYPE).
  LOAD('WEEDPROBLEM.HKB').
  DO(WEEDPROBLEM).
  REMOVE_TOPIC(WEEDPROBLEM)
END.

```

This code is abstracted from the actual code in WEEDPLUS to illustrate the memory organization implemented by using the REMOVE\_TOPIC() and LOAD() commands of the application language.

### **6.3 Increasing Knowledge Sharing and Re-use**

Because the PAD-BASED expert system utilizes the concepts of prototype and delegation, it fully supports knowledge sharing by its structure. In addition, the delegation mechanism also increases the flexibility of knowledge sharing among the prototypes.

In the object-oriented programming, reusability is defined as self-sufficiency of an object which enables it to be used independently and repeatedly. In the PAD-BASED system, each prototype is also self-defined, because it contains knowledge about the problem, basic behaviors such as problem-solving methodology, and the interfaces to the other prototypes that provide shared knowledge/operations to this prototype. These properties imply the self-sufficiency of the prototype for reusability. Although each prototype cannot be used independently in the sense that it requires the shared knowledge from the other prototypes, a group of prototypes consisting of the delegator and delegates can be used independently and repeatedly.

In the following two sections, knowledge sharing and re-use is described from two perspectives: internal knowledge sharing and re-use (in a PAD-BASED expert system), and external knowledge sharing and re-use (between a PAD-BASED expert system and a structured expert system).

#### **6.3.1 Internal Knowledge Sharing and Re-use**

Internal knowledge sharing and re-use in a PAD-BASED expert system is explicit. As an example, WEEDPLUS is designed to be used individually to provide recommendations for alfalfa growers and to provide educational information on a set of specific weeds (in Oklahoma) and a set of herbicides. The internal knowledge sharing

among the prototypes in WEEDPLUS is illustrated in figure 18. First, the WP\_PROTOTYPE is designed to share knowledge about weed problems for both the seedling stands (SL\_PROTOTYPE) and the established stands (ES\_PROTOTYPE). Because WP\_PROTOTYPE contains knowledge about weed problems for both situations, SL\_PROTOTYPE and ES\_PROTOTYPE can delegate to WP\_PROTOTYPE to activate the proper method. Second, WP\_PROTOTYPE can delegate to either WI\_PROTOTYPE to use its knowledge about specific weed type or to HI\_PROTOTYPE to use its knowledge about the recommended herbicide for controlling that particular weed. WI\_PROTOTYPE and HI\_PROTOTYPE can be used independently to provide knowledge about Oklahoma weeds and herbicides controlling alfalfa weeds directly by the users via MAIN\_PROTOTYPE.

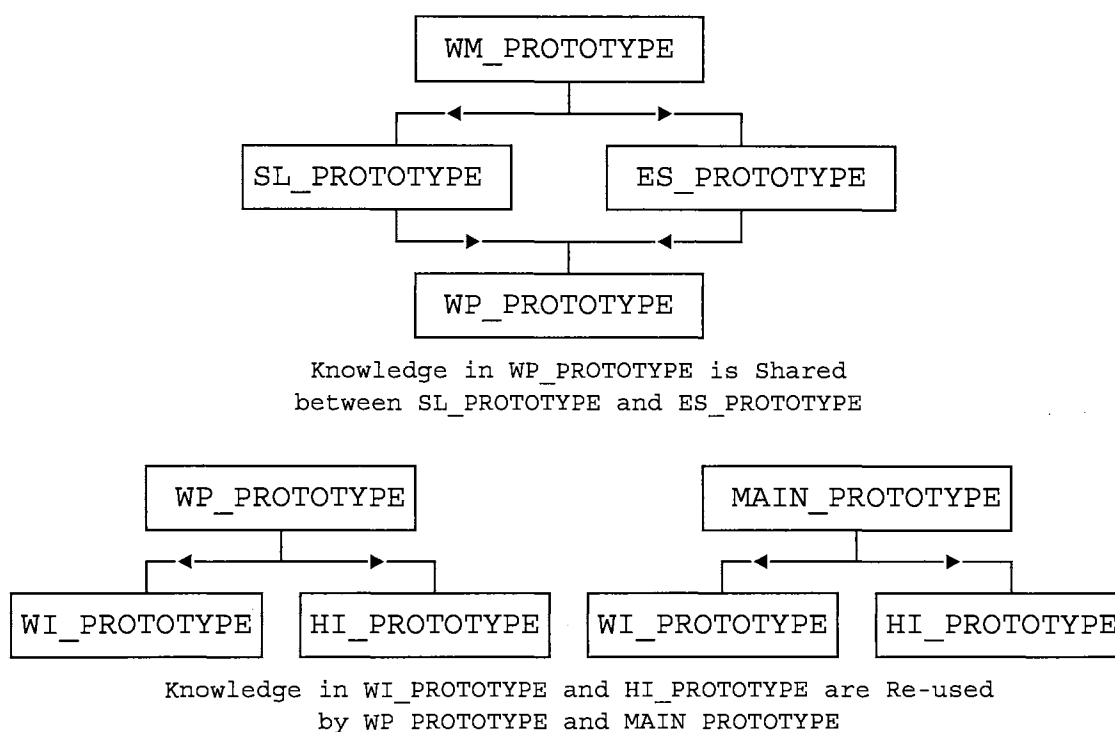


Figure 18. Internal Knowledge Sharing and Re-use in WEEDPLUS

To increase knowledge sharing and re-use, WEEDPLUS can be easily extended to share the knowledge on weed management, weed information, and herbicide information. The new prototypes containing new knowledge or knowledge that is different from knowledge in WEEDPLUS are created and interfaced to WEEDPLUS via delegation mechanism.

For instance, to use WEEDPLUS as an alfalfa weed management expert system for alfalfa growers in Indiana, Indiana weeds prototype (INWD\_PROTOTYPE) which contains Indiana weed information is added. The farmer *re-uses* the same weed management knowledge in WM\_PROTOTYPE (WEED MANAGEMENT) but WM\_PROTOTYPE delegates to the appropriate weed information based on what state the farmer specified (Oklahoma or Indiana). Figure 19 shows the modification of TASK 2 (in figure 12) after the information on Indiana weeds is added. Only the interface components of WI\_PROTOTYPE (WEED INFORMATION) are modified to provide a path to delegate to INWD\_PROTOTYPE (Indiana weeds prototype).

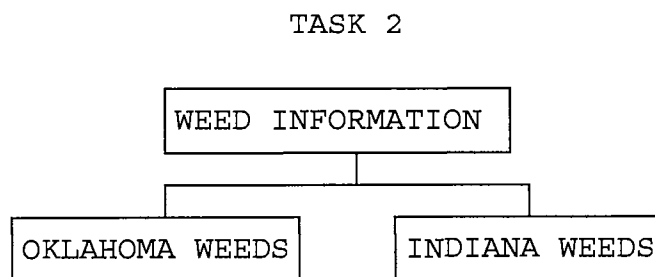


Figure 19. Modification of TASK 2 after Adding INDIANA Weeds

Figure 20 demonstrates the prototypes and delegation construction after Indiana weeds prototype is added. The Indiana farmers re-use the knowledge about weed

management for seedling and established stands, and knowledge on weed problem in WP\_PROTOTYPE. If the farmer indicates that STATE\_INFO is Indiana, WP\_PROTOTYPE delegates to WI\_PROTOTYPE which in turn delegates to INWD\_PROTOTYPE to provide knowledge about Indiana weeds. Otherwise, WI\_PROTOTYPE delegates to OKWD\_PROTOTYPE for Oklahoma weeds.

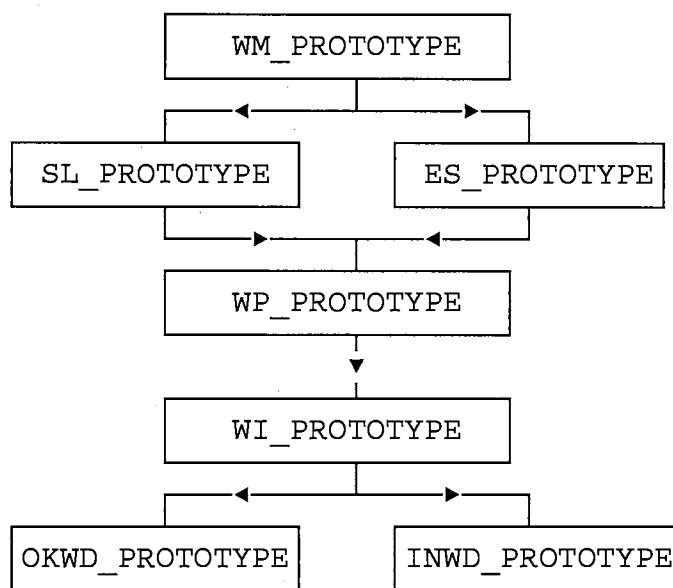


Figure 20. Prototypes Construction after Adding Indiana Weeds Prototype; Indiana Farmers Re-use Knowledge in WEEDPLUS

Typically, to add new prototypes for knowledge sharing and re-use, the interface section of the delegator plays the most important role. In the above example, interface sections of WP\_PROTOTYPE and WI\_PROTOTYPE are modified to provide the proper delegation paths. WP\_PROTOTYPE is modified to add a delegation path to WI\_PROTOTYPE (DELEGATE\_TO\_WEEDUNIT) while WI\_PROTOTYPE needs to have a path to INWD\_PROTOTYPE (DELEGATE\_TO\_INWEED). Figure 21 shows the structure of the WP\_PROTOTYPE and WI\_PROTOTYPE after modifying their interfaces.



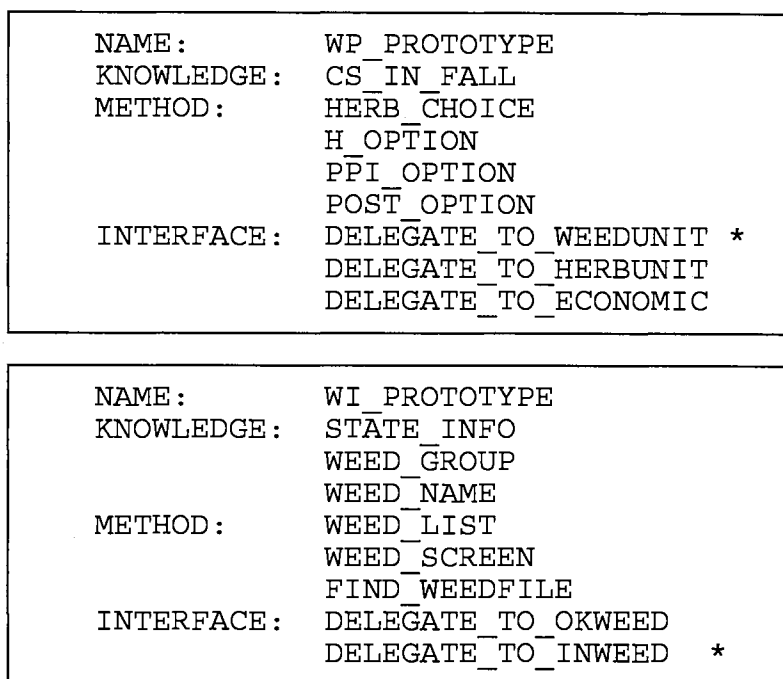


Figure 21. Modification in the Interface after Adding a New Prototype

Figure 22 displays the delegation mechanism after INWD\_PROTOTYPE is added.

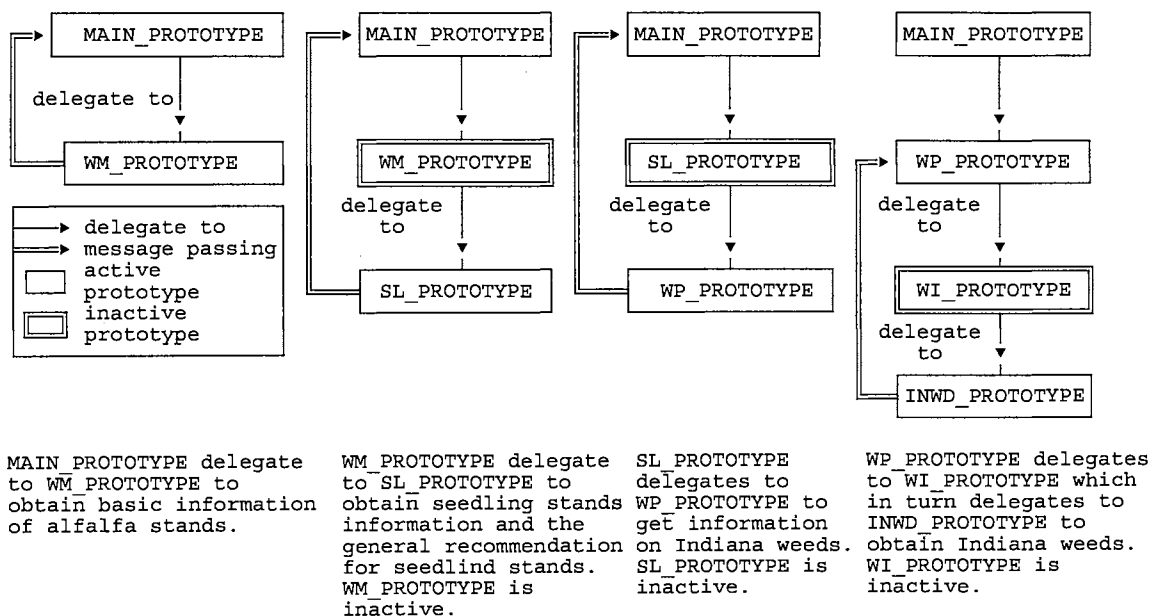


Figure 22. Knowledge Re-use in WEEDPLUS Via Delegation Mechanism

Indiana farmers re-use knowledge in WEEDPLUS on the weed management (WM\_PROTOTYPE) in seedling stands (SL\_PROTOTYPE). They also re-use knowledge about weed problems (WP\_PROTOTYPE). The new knowledge on Indiana weeds is accessible via the delegation path provided in the weed unit (WI\_PROTOTYPE).

The above example demonstrates that knowledge organized in the PAD-BASED structure fully supports internal knowledge sharing and re-use. Conceptually, only the interface section of the prototype needs to be modified when a new prototype is added for knowledge sharing and re-use.

### **6.3.2 External Knowledge Sharing and Re-use**

In this section, the external knowledge sharing and re-use is described in two aspects: 1) between a PAD-BASED expert system and a structured existing expert system (incompatible structure) and 2) between two PAD-BASED expert systems (compatible structure). This advantage is demonstrated by using WEEDPLUS and PROFALF.

#### **6.3.2.1 PROFALF--an Expert System to Estimate Profitability of Alfalfa.**

PROFALF is an expert system used as an educational tool for extension people and the alfalfa growers. It is designed to incorporate detailed information about alfalfa production regarding site, soil, machinery, management levels, and yield information to simulate the annual costs and returns over the alfalfa projected stand life. The system requires information from several domains of expertise such as Agronomy, Entomology, Agricultural Engineering, and Agricultural Economy. It generates a soil fertility profile from the site information and projects the annual alfalfa yields based on the user's fertility and liming plans. It projects the annual weed and insect control costs based on

typical systems plan.

PROFALF is one of the expert systems in Alfalfa Integrated Management (AIM) system which consists of four units: ALFWEEV, HAYMACH\$, PROFALF, and WEEDPLUS as displayed in Figure 23. Each unit of AIM system is a stand-alone system. ALFWEEV is the alfalfa weevil insect management expert system designed to provide recommendations on insect management cost and practice. HAYMACH\$ is a hay harvesting and machinery expert system providing a cost analysis of specific hay harvesting operation (cutting, raking and baling). PROFALF [Limsupavanich et al. 94] is developed in a hybrid structure of shell-based (FIKES and KEHLER, 1985), hypertext (CONKLIN, 1987), and conventional subprogram system (MARAN and BECK, 1989).

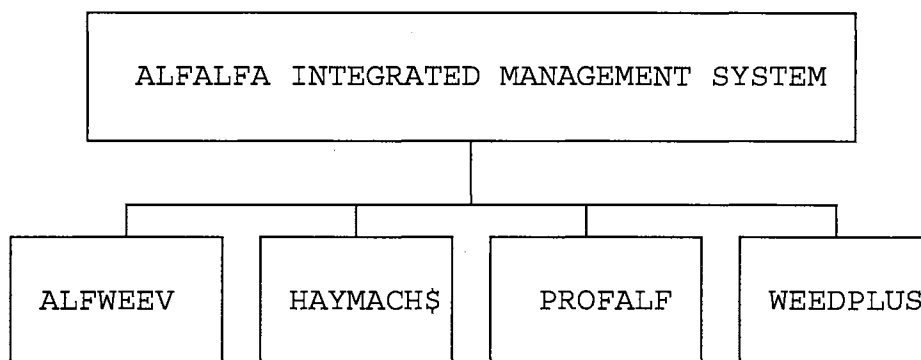


Figure 23. Components of the AIM System

Figure 24 illustrates PROFALF schematic diagram. It is also developed in the KnowledgePro® environment. It manipulates the facts and the knowledge base via the expert system shell; and provides explanations, recommendations, and results via the hypertext system. Figure 25 shows the subsystem organization in PROFALF.

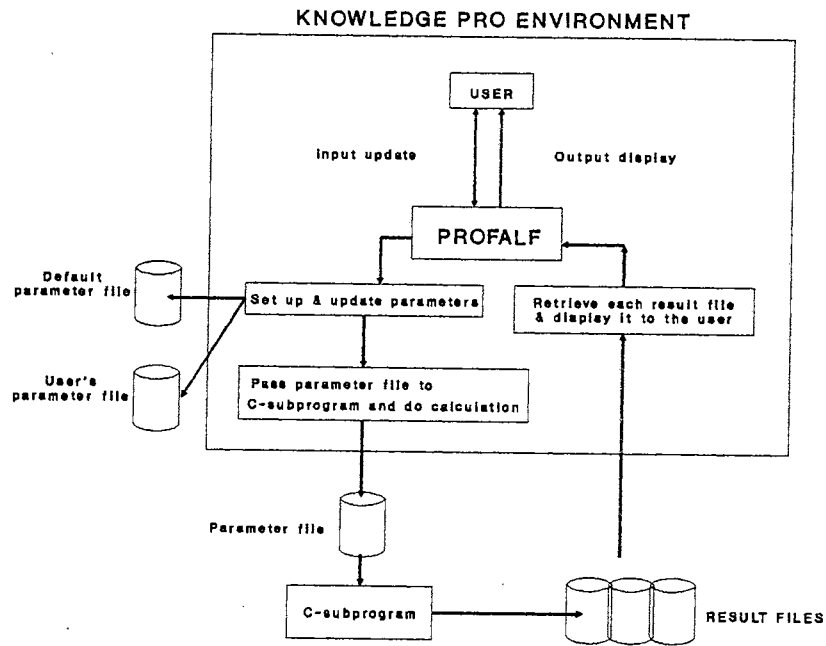


Figure 24. PROFALF Schematic Diagram

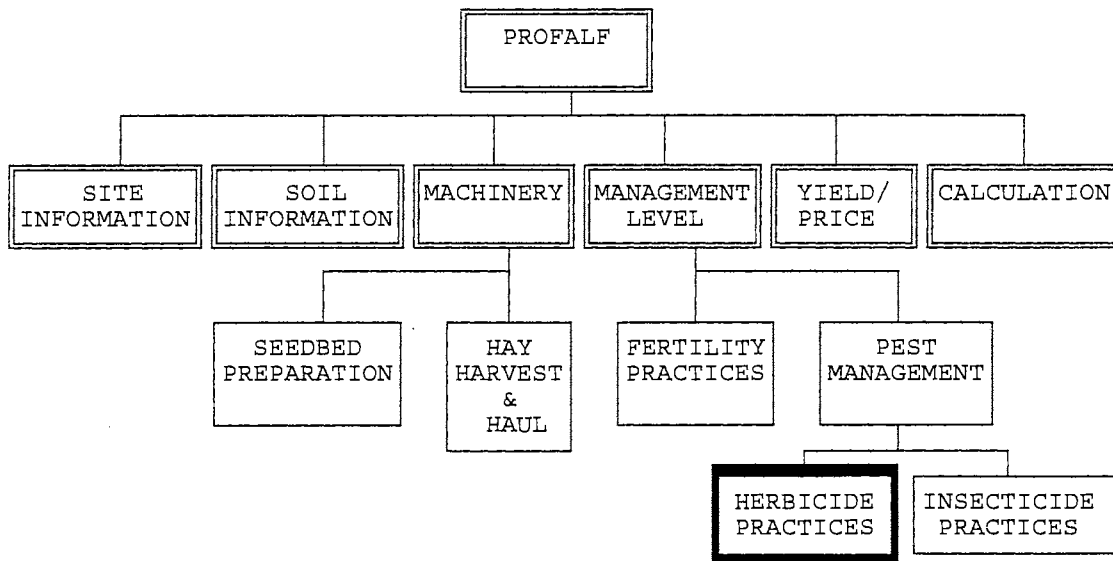


Figure 25. Subsystem Organization of PROFALF

**6.3.2.2 Incompatible Structure.** The knowledge sharing and re-use between two expert systems with incompatible or different structures involves a PAD-BASED expert system (WEEDPLUS) and a non-PAD-BASED expert system (PROFALF). In particular, a non-PAD-BASED expert system must be designed and developed in a highly structured fashion. For simplicity, two expert systems which are created by the same developing tool will be used. The knowledge in WEEDPLUS can be re-used in the task herbicide practices of PROFALF (see figure 25). Conceptually, both expert systems are treated as two big prototypes. Despite its knowledge organization, PROFALF is a prototype with its interface section missing. Figure 26 shows the conceptual structure to provide a bridge for PROFALF to re-use knowledge in WEEDPLUS. To create the bridge for this knowledge sharing and re-use, the *delegator prototype* is added to define the paths for delegation and message passing between PROFALF and WEEDPLUS. In other words, PROFALF activates the delegator prototype to *delegate to* WEEDPLUS to *re-use* its knowledge. Then the result is sent back to PROFALF via the message passing mechanism.

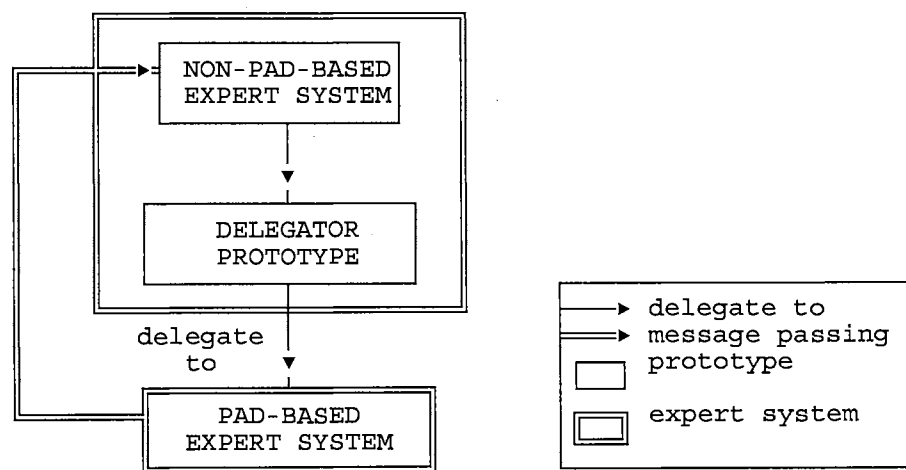


Figure 26. Conceptual Idea for Non PAD-BASED Expert System to Re-use Knowledge from a PAD-BASED Expert System

In fact, the delegator prototype serves as the interface unit for PROFALF. The structure of this delegator prototype (X\_PROTOTYPE) is elaborated in figure 27. It contains information from PROFALF, information about what is needed and where to incorporate new knowledge to PROFALF. Its basic behaviors consist of STORE\_INPUT, SAVE\_OUTPUT, and RETRIEVE\_OUTPUT. Its interface section provides the delegation path to WEEDPLUS, and the interface to input and output information file. These input and output information files are used as a working space or blackboard for the message passing mechanism.

NAME:	X_PROTOTYPE
KNOWLEDGE:	information from PROFALF to WEEDPLUS information needed from WEEDPLUS where to incorporate new knowledge to PROFALF
INTERFACES:	delegate to WEEDPLUS interface to input information file interface to output information file

Figure 27. The General Structure of the Delegator Prototype

To treat PROFALF as a prototype, consider that PROFALF does not have the delegation path to the new prototype (X\_PROTOTYPE). In this case, another path must be added into PEST MANAGEMENT UNIT in PROFALF as a delegation path to access the X\_PROTOTYPE. This modification requires the high structurability in PROFALF, otherwise it would be more complicated to change something in the existing expert system without interfering with its functionality..

The delegation mechanism from PROFALF to WEEDPLUS can be designed in two levels: macro and micro levels. *Macro level* means that PROFALF treats WEEDPLUS as one uniform prototype and utilizes the delegation paths that have already been designed in WEEDPLUS to provide knowledge sharing and re-use. As illustrated

in figure 26, X\_PROTOTYPE delegates to MAIN\_PROTOTYPE in WEEDPLUS. *Micro level* means that the delegator prototype (X\_PROTOTYPE) can be designed to contain all the major delegation paths to the prototypes in WEEDPLUS that PROFALF needs to obtain for shared or re-used information. However, in this case macro level is recommended since PROFALF can re-use most of the knowledge in WEEDPLUS. With incompatible structure, the micro level is possible in one way, i.e., from a non-PAD-BASED expert system to re-use knowledge in a PAD-BASED expert system. In the opposite direction, knowledge sharing and re-use in the micro level from a PAD-BASED expert system to a non-PAD-BASED expert system is impossible because there are no delegation paths provided.

**6.3.2.3 Compatible Structure.** Knowledge sharing and re-use between two PAD-BASED expert systems can be designed in both macro and micro levels because of its compatible structure. In addition, knowledge sharing and re-use can be fulfilled in both ways. First, in the macro level the same concept as in section 6.3.2.1 is applied. A delegator prototype is defined to join two PAD-BASED expert systems. Each expert system is treated as a big prototype to perform one big task. The conceptual idea is demonstrated in figure 28.

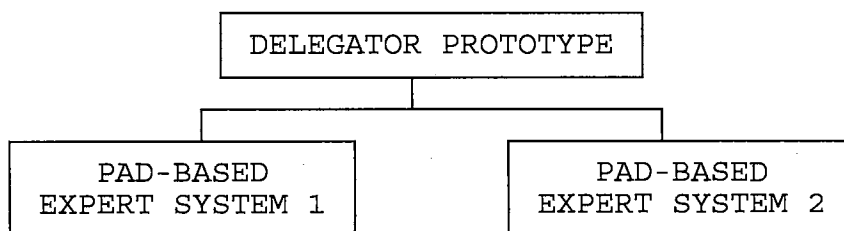


Figure 28. Knowledge Sharing and Re-use between Two PAD-BASED Expert Systems in the Macro Level

The delegator prototype has the same structure as X\_PROTOTYPE in the previous section. It also utilizes the information file for message passing among two expert systems. Knowledge sharing and re-use is accomplished via the delegation paths in both expert systems.

In the micro level, for instance, if TASK 1 in the first PAD-BASED expert system wanted to re-use the operation in SUBTASK 3.1 of the second PAD-BASED expert system in figure 29, a new prototype (Y\_PROTOTYPE) can be created and added directly to delegate to the SUBTASK 3.1 prototype. This augmentation concept helps keep the modification in both expert systems minimal. The function of this new augmented prototype will not interfere with the existing functionality of both expert systems. The modification is limited to the interface section of TASK 1 in the first expert system to provide delegation path to the Y\_PROTOTYPE.

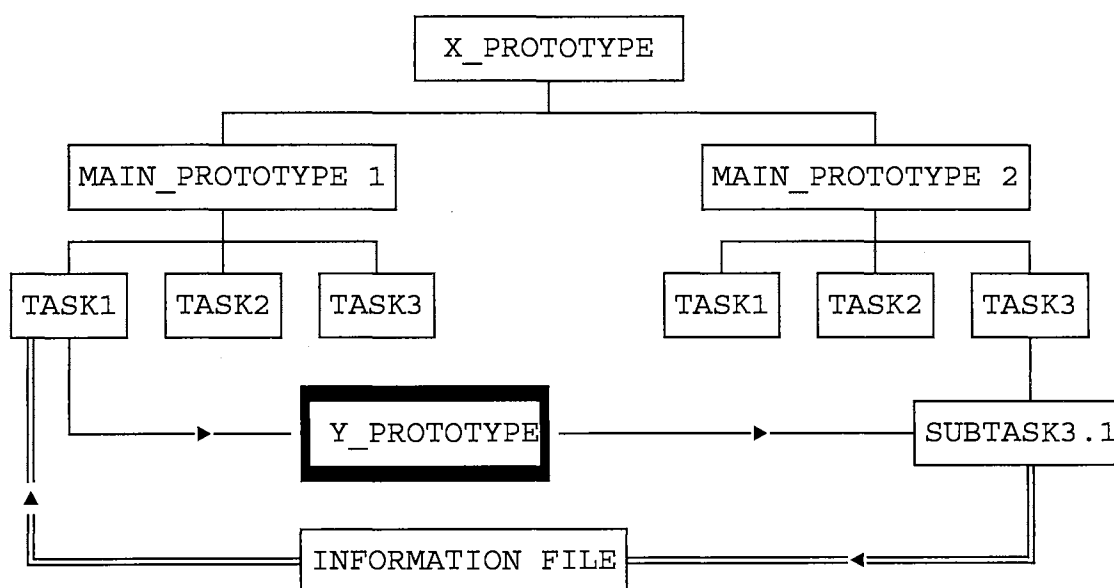


Figure 29. Knowledge Sharing and Re-use in the Micro Level between Two PAD-BASED Expert Systems



## CHAPTER VII

### CONCLUSIONS AND FUTURE WORK

Object-oriented approaches are dominant development methods in this decade. In this dissertation, the studies of these approaches applied to expert systems are presented. The fundamental concepts, advantages and disadvantages of class/subclass and prototypes are reviewed. In addition, the descriptions of three mechanisms: inheritance, delegation, and polymorphism are pointed out. The research shows that the concepts of class/subclass via inheritance and polymorphism mechanisms can be applied successfully to expert system design and maintenance. However, delegation mechanism still remains unexplored.

In response to the above study and the problems confronted in designing and developing the expert systems in agricultural applications, it brings about a new expert system designing model called the PAD-BASED model. This model utilizes the concepts of prototypes (without class/subclass hierarchy) and the delegation mechanism to provide the knowledge organization in expert system design. The description of the PAD-BASED model, its architecture, applications, and advantages have been described.

In summary, the PAD-BASED model is beneficial to expert system design and development. There are five major contributions. First, the PAD-BASED expert system contains a high modular structure. This benefit is obtained directly from the property of

its organization as a collection of prototypes. Second, the PAD-BASED development life cycle model provides a means to the knowledge acquisition on each subtask in the task/subtask organization. In addition, it provides a parallel environment for the experts and the knowledge engineers in the knowledge acquisition, prototypes and interfaces construction, and prototype testing steps. This parallel environment directly helps reduce the development time and generate a high quality outcome. Third, the PAD-BASED model provides an alternate memory management scheme to remedy the limited internal memory problem in small computers. This problem is the factor to limit the expert system distribution among the general users who gain access to small PCs. In this scheme, the delegation mechanism plays the most important role because only the original delegator remains active in the main memory. This space benefit is the most important advantage of the delegation mechanism as compared to the speed benefit that inheritance provides. Fourth, knowledge sharing and re-use among the PAD-BASED expert systems are fully supported. However, only a non PAD-BASED expert system can re-use sharing knowledge from a PAD-BASED expert system. Finally, maintenance and extension of the PAD-BASED expert system are simple, since another prototype can be designed and interfaced to the existing system without interfering with its functionality.

The above contributions show that the PAD-BASED model is a simple, but powerful, technique in designing and maintaining expert systems. Furthermore, the PAD-BASED model is generally applicable, although the emphasis of this dissertation is on the agricultural applications. With the PAD-BASED model, other areas such as machine

simulation, modeling, manufacturing systems, and diagnostic system could be applied. Another research opportunity is to use the PAD-BASED model as a maintaining model for the non-structured existing expert systems to generate a new expert system in the PAD-BASED structure. This research would be another major contribution to reduce the complexity in expert system maintenance.

## BIBLIOGRAPHY

- [Aksit et al. 91] M. Aksit, J.M. Dijkstra, and A. Tripathi, "Atomic Delegation: Object-Oriented Transactions," *IEEE Software*, Vol. 8, No. 2, March 1991, pp. 84-92.
- [Almarode 89] J. Almarode, "Rule-based Delegation for Prototypes," *OOPSLA '89 Proceedings*, ACM, New York, October, 1989, pp. 363-370.
- [Alpert et al. 90] S.R. Alpert, S.W. Woyak, H.J. Shrobe, and L.F. Arrowood, "Object-Oriented Programming in AI," *IEEE Expert*, Vol. 5, No. 6, December, 1990, pp. 6-7.
- [Bapa Rao et al. 89] K.V. Bapa Rao, A. Gafni, and G. Raeder, "Dynamo: A Model for a Distributed Multi-media Information Processing Environment," *IEEE Proceedings of the Hawaii International Conference on System Science*, V. II, 1989, pp. 800-809.
- [Baran 89] N. Baran, "The Loneliness of the Low-Budget User," *BYTE*, Vol. 14, No. 8, August, 1989, pp. 344.
- [Barghouti and Kaiser 90] N.S. Barghouti, and G.E. Kaiser, "An Object-Oriented Framework for Modeling Cooperation in Multi-Agent Rule-Based Development Environments," *Technical Report CUCS-017-90*, Department of Computer Science, Columbia University, New York, NY, May, 1990.
- [Bertolucci 90] J. Bertolucci, "The 640-byte Solution?," *BYTE*, Vol. 15, No. 3, March 1990, pp. 208-214.
- [Blair et al. 89] G.S. Blair, J.J. Gallagher, and J. Malik, "Genericity vs. Inheritance vs., Delegation vs. Conformance vs....," *J. Object-Oriented Programming*, Vol. 2, No. 3, Sept./Oct. 1989, pp. 11-17.
- [Bobrow and Stefik 83] D.G. Bobrow, and M. Stefik, *The LOOPS Manual*, Xerox Palo Alto Research Center, December, 1983.
- [Boehm 81] B.W. Boehm, *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, NJ, 1981.

- [**Borning 86**] A. Borning, "Classes Versus Prototypes in Object-Oriented Languages," *Fall Joint Computer Conference*, ACM/IEEE, Dallas, Texas, November, 1986.
- [**Cardelli and Wegner 85**] L. Cardelli and P. Wegner, "On Understanding Types, Data Abstraction, and Polymorphism," *ACM Computing Surveys*, Vol. 17, No. 4, December, 1985, pp. 471-522.
- [**Chandrasekaran, et al. 92**] B. Chandrasekaran, T.R. Johnson, and J.W. Smith, "Task-Structure Analysis for Knowledge Modeling," *CACM*, Vol. 35, No. 9, September, 1992, pp. 124-137.
- [**Cholawski 88**] E.M. Cholawski, "Beating the Prototype Blues," *AI Expert*, December, 1988.
- [**Corradi and Leonardi 91**] A. Corradi, and L. Leonardi, "PO Constraints as Tools to Synchronize Active Objects," *J. Object-Oriented Programming*, Vol. 4, No. 6, October, 1991, pp. 41-53.
- [**Coulson et al. 89**] R.N. Coulson, M.C. Saunders, D.K. Loh, F.L. Oliveria, D. Drummond, P.J. Barray, and K.M. Swain, "Knowledge System Environment for Integrated Pest Management in Forest Landscapes: The Souther Pine Beetle (Coleoptera: Scolytidae)," *Bulletin of the ESA*, American Entomologist, Summer, 1989, pp. 26-32.
- [**Fikes and Kehler 85**] R. Fikes, and T. Kehler, "The Role of Frame-Based Representation in Reasoning," *CACM*, Vol. 28, No. 9, September, 1985, pp. 904-920.
- [**Franke 90**] D. W. Franke, "Imbedding Rule Inferencing in Applications," *IEEE Expert*, Vol. 5, No. 6, December, 1990, pp. 8-14.
- [**Freeman-Benson 90**] B.N. Freeman-Benson, "Kaleidoscope: Mixing Objects, Constraints, and Imperative Programming," *ECOOP/OOPSLA '90 Proceedings*, October, 1990, pp.77-88.
- [**Horn 92**] B. Horn, "Constraint Patterns as a Basis for Object Oriented Programming," *OOPSLA '92 Proceedings*, 1992, pp. 218-233.
- [**Huhnke et al. 90**] R. Huhnke, W. Bowers, J.A. Stark, and J. Limsupavanich, "*HAYMACH\$*: Hay Equipment Cost Estimation Software," Oklahoma State University, Cooperative Extension Service, Computer Software Series, CSS-50, 1990.
- [**Ibrahim and Woyak 90**] M.H. Ibrahim, and S.W. Woyak, "An Object-Oriented Environment for Multiple AI Paradigms," *Proceeding of IEEE International Conference Tools for Artificial Intelligence*, 1990, pp. 77-83.

- [**Johnson and Nofziger 90**] G.V. Johnson and D.L. Nofziger, "*NPK\$PLUS: An Interactive Microcomputer Program to Interpret Soil Test Results and to Evaluate the Economics of Alternative Fertilizer Application Rates*," Oklahoma State University, Cooperative Extension Service, Computer Software Series, CSS-47, 1990.
- [**Johnson and Zweig 91**] R.E. Johnson, and J.M. Zweig, "Delegation in C++," *J. Object-Oriented Programming*, Vol. 4, No. 7, November/December, 1991, pp. 31-34.
- [**Kannan and Dodrill 90**] R. Kannan, and W.H. Dodrill, "DAIS: A Distributed AI Programming Shell," *IEEE Expert*, Vol. 5, No. 6, December, 1990, pp. 34-42.
- [**Kehler and Clemenson 84**] T.P. Kehler and G.D. Clemenson, "An Application Development System for Expert Systems," *Syst. Softw.*, Vol. 3, No. 1, January, 1984, pp. 212-224.
- [**Klahr and Waterman 86**] P. Klahr and D.A. Waterman, *Expert Systems Techniques, Tools, and Application*, Addison-Wesley, Reading, MA, 1986
- [**Leonardi et al. 89**] L. Leonardi, P. Mello, and A. Natali, "Prototypes in Prolog," *J. Object-Oriented Programming*, Vol. 2, No. 3, September/October, 1989, pp.20-28.
- [**Leung and Wong 90**] K.S. Leung, and M.H. Wong, "An Expert-System Shell Using Structured Knowledge: An Object-Oriented Approach," *Computer*, Vol. 23, No. 3, March, 1990, pp. 38-47.
- [**Lieberman and Hewitt 83**] H. Lieberman, and C. Hewitt, "A Real Time Garbage Collector Based on the Lifetimes of Objects," *CACM*, Vol. 26, No. 6, June, 1983.
- [**Lieberman 86a**] H. Lieberman, "Using Prototypical Objects to Implement Shared Behavior in Object Oriented System," *OOPSLA '86 Proceedings*, September 1986, pp. 214-223.
- [**Lieberman 86b**] H. Lieberman, "Delegation and Inheritance: Two Mechanisms for Sharing Knowledge in Object Oriented Systems," J. Bezivin, P. Cointe (editors), *3eme Journees d'Etudes Languages Orientes Objets*, AFCET, Paris, France, 1986, pp. 79-89.
- [**Lieberman 87**] H. Lieberman, "Languages, Object Oriented," *Encyclopedia of Artificial Intelligence*, Vol. 1, S.C. Shapiro, ed., John Wiley & Sons, New York, 1987, pp. 452-456.

- [**Limsupavanich et al. 92**] J. Limsupavanich, J.A. Stark, G.W. Cuperus, C. Ward, G. Johnson, R. Huhnke, J. Stritzke, and R. Berberet, "*PROFALF: Profitability of Alfalfa*," Oklahoma State University, Cooperative Extension Service, Computer Software Series, CSS-51, 1992.
- [**Limsupavanich et al. 94**] J. Limsupavanich, J.A. Stark, G.W. Cuperus, C. Ward, G. Johnson, R. Huhnke, J. Stritzke, and R. Berberet, "An Expert System for Estimating of Alfalfa Profitability," to be presented and published in the *Proceedings of the 5th International Conference on Computers in Agriculture*, ASAE, Orlando, Florida, February, 1994.
- [**McGraw and Harbison-Briggs 89**] K.L. McGraw and K. Harbison-Briggs, *Knowledge Acquisition: Principles and Guidelines*, Printice Hall, Englewood Cliffs, NJ, 1989.
- [**Malloy 89**] R. Malloy, "VROOMM Goes the Spreadsheet," *BYTE*, Vol. 14, No. 10, October, 1989, pp. 111-112.
- [**Martin and Oxman 88**] J. Martin, and S. Oxman, *Building Expert Systems*, Prentice Hall, Englewood Cliffs, N.J., 1988.
- [**Mitrpanont et al. 93**] J.L. Mitrpanont, J. Stritzke, and G.W. Cuperus, "*WEEDPLUS: Weed Management Expert System*," Oklahoma State University, Cooperative Extension Service, Computer Software Series, CSS (In Press), 1993.
- [**Mitrpanont et al. 94**] J.L. Mitrpanont, K.M. George, and P. Benjamin, "Prototype and Delegation-Based Approach to Knowledge Organization in Expert System Design," to be presented and published in the Computer Applications Symposium Session on Knowledge-Based Systems in Engineering, Energy-Sources Technology Conference & Exhibition (ETCE), *ASME*, New Orleans, LA, January, 1994.
- [**Mitrpanont et al. 94**] J.L. Mitrpanont, K.M. George, and G.W. Cuperus, "PAD-BASED Expert System in Agricultural Applications," *to be published*.
- [**Mitrpanont et al. 94**] J.L. Mitrpanont, K.M. George, and G.W. Cuperus, "PAD-BASED Expert System in Small Computer System," to be presented and published in the *Proceedings of 1994 Symposium on Applied Computing (SAC'94)*, Phoenix, Arizona, March, 1994.
- [**Narayanan and Jin 91**] A. Narayanan, and Y. Jin, "An Object-Oriented Approach to Expert Diagnostic Systems," *J. Object-Oriented Programming*, Vol. 4, No. 6, October, 1991, pp. 19-29.
- [**Naegele et al. 85**] J.A. Naegele, R.N. Coulson, N.D. Stone, and R.E. Frisbie, "The Use of Expert Systems to Integrate and Deliver IPM Technology," *CIPM Integrated Pest Management on Major Agricultural Systems*, Texas A&M University, 1985, pp. 692-710.

- [Noll and Scacchi 91] J. Noll, and W. Scacchi, "Integrating Diverse Information Repositories: A Distributed Hypertext Approach," *Computer*, Vol. 24, No. 12, December, 1991, pp. 38-45.
- [Park 91] H. Park, "Abstract Object Types=Abstract Knowledge Types+Abstract Data Types+Abstract Connector Types," *J. Object-Oriented Programming*, Vol. 4, No. 3, June, 1991, pp. 37-52.
- [Ping et al. 90] C. Ping, C. Xiyao, and J. Yimin, "An Approach to Introduce the Reflection to C++," *Proceedings of the 14th Annual International Computer Software and Applications Conference-COMPSAC 90*, 1990, pp. 52-56.
- [Prerau et al. 91] D.S. Prerau, A.S. Gunderson, R.E. Reinke, and Adler, M.R., "Maintainability Techniques in Developing Large Expert System," *IEEE Expert*, Vol. 6, No. 3, June, 1991, pp. 71-80.
- [Ramamoorthy and Sheu 88] C.V. Ramamoorthy, and P.C. Sheu, "Object-Oriented Systems," *IEEE Expert*, Vol. 3, No. 3, Fall 1988, pp. 9-15.
- [Redin 87] P. Redin, "Developing ES on PC's--A Methodology," *AI Expert*, October, 1987.
- [Rhykerd et al. 92] L.M. Rhykerd, R.L. Rhykerd, B.A. Engel, and C.L. Rhykerd, "Use of Knowledge Engineering to Maximize Forage Production of MEDICAGO SATIVA L.," *Proceedings of the 4th International Conference on Computers in Agriculture*, ASAE, Orlando, Florida, January, 1992, pp. 165-170.
- [Royce 70] W.W. Royce, "Managing the Development of Large Software Systems: Concepts and Techniques," *Proceedings of WESCON*, August, 1970.
- [Schwartz 87] T. Schwartz, "PC Perspectives," *IEEE Expert*, Vol. 2, No. 3, FALL 1987, pp. 80-84.
- [Stark et al. 89] J.A. Stark, J. Stritzke, and G.W. Cuperus, "*WEEDALF: An Alfalfa Weed Management Software*," Oklahoma State University, Cooperative Extension Service, Computer Software Series, CSS-43, 1989.
- [Stefik and Bobrow 86] M. Stefik, and D.G. Bobrow, "Object-Oriented Programming: Themes and Variations," *AI Magazine*, Vol. 6, No. 1, Winter, 1986, pp. 40-62.
- [Stein 87] L.A. Stein, "Delegation is Inheritance," *ACM Conference Proceedings OOPSLA '87*, Orlando, Special Issue, SigPlan Notices, Vol. 22, No. 12, December, 1987, pp. 138-146.



- [Tomlinson et al. 89] C. Tomlinson, M. Scheevel, and W. Kim, "Sharing and Organization Protocols in Object-Oriented Systems," *J. Object-Oriented Programming*, Vol. 2, No. 4, November/December, 1989, pp. 25-36.
- [Turban 92] E. Turban, *Expert Systems and Applied Artificial Intelligence*, Macmillan, New York, NY, 1992.
- [Ungar and Smith 87] D. Ungar, and R.B. Smith, "Self: The Power of Simplicity," *ACM Conference Proceedings OOPSLA '87*, Orlando, pp. 227-242.
- [van den Bos and Laffra 91] J. van den Bos, and C. Laffra, "PROCOL. A Concurrent Object-Oriented Language with Protocols Delegation and Constraints," *Acta Informatica*, Vol. 28, No. 6, Jan., 1991, pp. 511-538.
- [Wegner 87] H. Wegner, "Dimensions of Object-Based Language Design," *OOPSLA '87 Proceedings*, ACM, Orlando, 1987, pp. 168-182.
- [Yemini et al. 91] Y. Yemini, G. Goldszmidt, and S. Yemini, "Network Management by Delegation," *Integrated Network Management*, II, I. Krishman and W. Zimmer (Editors), IFIP, 1991, pp. 95-107.
- [Yen et al. 91] J. Yen, R. Neches, and R. MacGregor, "Clasp: Integrating Term Subsumption Systems and Production Systems," *IEEE Trans. Knowledge and Data End.*, Vol. 3, No. 1, March, 1991, pp. 25-32.
- [Yen et al. 91] J. Yen, H. Juang, and R. Macgregor, "Using Polymorphism to Improve Expert System Maintainability," *IEEE Expert*, Vol. 6, No. 2, April, 1991, pp. 48-55.
- [Zhu et al. 91] J. Zhu, R. Nassif, P. Goyal, and R. Mikkilineni, "A Perspective on Object-Oriented Technology," *36th IEEE Computer Society International Conference - COMPCON Spring 1991*, pp. 546-552.
- [Zucker 89] J. Zucker, *Engineering Design Computed by Prototypes and Descriptions*, Ph.D. Dissertation, Open University, United Kingdom, 1989.

## GLOSSARY

**Backward Chaining:** a goal-driven reasoning technique in which search continues working *backward* through successive subgoals until it works back to the facts of the problem.

**Class:** the description of a group of objects that have common characteristics and behaviors/methods.

**Decision Support System (DSS):** the computer-based information system that is used to provide knowledge or information in supporting decision making or solving the nonstructured problems.

**Delegation:** a mechanism that allows prototype to delegate responsibility to another prototype to perform the task or operations for the delegator.

**Domain Expert:** a person with expertise in the domain in which the expert system is being designed and developed.

**Forward Chaining:** a data-driven search scheme in which the search begins with the facts of a problem and proceeds by applying rules to the solution.

**Frame:** a knowledge representation scheme that organizes knowledge in various slots and utilizes the class/subclass concepts and the inheritance mechanism.

**Hybrid Environment:** a software developing environment that is used to develop an expert system and provides several knowledge representation schemes.

**Hypertext:** an approach to manipulate text or information by utilizing nodes and links to allow an unstructured presentation.

**Inference Engine:** a mechanism that actually performs the deduction process or the reasoning process in an expert system.

**Inheritance:** a mechanism that allows sharing methods/operations among classes, subclasses, and objects.

**Integrated Pest Management (IPM):** a comprehensive, systematic approach to commodity protection that emphasizes increased information for improved decision making to reduce purchased inputs and optimize social, economic, and environmental consequences.

**Invoking:** a mechanism that initiates an operation of a prototype.

**Knowledge Engineering:** a term that describes the process of developing and maintaining expert systems which involves the cooperation of human experts and knowledge engineers to explicitly extract and implement the knowledge and methods that the human experts use to solve the problem.

**Object:** the primitive entity in the object-oriented programming which has a unique identity, contains a private data structure, and can be accessed or modified by its predefined methods upon receiving a proper message.

**The PAD-BASED Model:** an expert system development methodology that is based on the concepts of the prototypes and delegation mechanism.

**Polymorphism:** a mechanism that provides different interpretations of the same message when received by different objects.

**Prototype:** a concrete structure that is used to organize knowledge in the PAD-BASED expert system

**Rapid Prototyping:** an expert system development methodology for a rapid development of a part of an expert system to demonstrate its functionality and to meet user and manager requirements.

**Reusability:** a self-sufficiency property of an object which enables it to be used or re-used independently and repeatedly.

**Simple Message Passing:** a mechanism that provides information transferring among the prototypes

**Software Development Life Cycle:** a term generally used to describe the process of developing and maintaining software.

**Stand:** a term that is used to describe a growth of plants particularly soon after germination with regard to the distribution of the plants in a given area.

**Weed Management:** the integration of the various weed control options into a management system that is effective, environmentally sound, and profitable.

2  
VITA

Jarensri Limsupavanich-Mitrpanont

Candidate for the Degree of

Doctor of Philosophy

Dissertation: PAD-BASED: PROTOTYPES AND DELEGATION BASED APPROACH  
TO KNOWLEDGE ORGANIZATION IN EXPERT SYSTEM DESIGN

Major Field: Computer Science

Biographical:

Personal Data: Born in Bangkok, Thailand, September, 1958, the daughter of Chong-hung and Hoong-jaung Sae Lim.

Education: Graduated from Trium Udom Suksa High School, Bangkok, Thailand, March, 1976; received the Bachelor of Science degree in Physics from Mahidol University, Bangkok, Thailand, April, 1980; received the Master of Science degree in Applied Mathematics from Mahidol University, Bangkok, Thailand, November, 1983; completed the requirements for the Doctor of Philosophy degree at Oklahoma State University, December, 1993.

Professional Experience:

1980-1981: Programmer at the Office of Registrar, Academic Affair section at Mahidol University;

1983-1986: Software Developer at Mahidol University Computing Center designed and developed Mahidol University student database system including enrollment and grade report system. Lecturer at Mahidol University teaching programming languages and data processing courses to Medical Science students of both Ramathibodhi and Siriraj Hospitals.

1988-1993: Software Specialist, IPM Extension, Entomology Department, Oklahoma State University, designed and developed expert system series: HAYMACH\$ (received the ASAE Blue Ribbon Outstanding Award in 1991), PROFALF, WHEA\$CON, WEEDPLUS, and softwares for 4-H "Caring for Planet Earth" program: WASTEMAN and FORESTRY games (recognized as a model program of environmental excellence by the National Environmental Awards Council (NEAC) in 1993 and listed in the 4th edition of Renew America's Environmental Success Index). Ph.D. dissertation research was recognized as an honorable mention for the OSU Graduate Research Excellent Award for Fall 1993.